

AD-A208 378

RADC-TR-88-324, Vol II (of nine), Part B
Interim Report
March 1989



NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 Discussing, Using, and Recognizing Plans

Syracuse University

Stuart C. Shapiro and Beverly Woolf

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

A handwritten signature or set of initials, possibly "AS", is written in ink to the right of the "APPROVED FOR PUBLIC RELEASE" text.

**ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700**

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-324, Vol II (of nine), Part B has been reviewed and is approved for publication.

APPROVED:



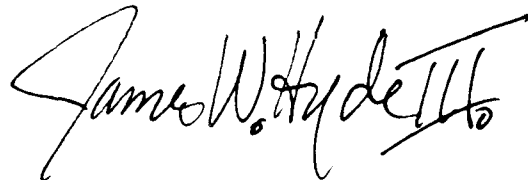
ROBERT L. RUSSEL, JR., Capt, USAF
Project Engineer

APPROVED:



WALTER J. SENUS
Technical Director
Directorate of Intelligence & Reconnaissance

FOR THE COMMANDER:



JAMES W. HYDE, III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDP) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-324, Vol II (of nine), Part B		
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A			7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (IRDP)		
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC)		6b. OFFICE SYMBOL (If applicable)		7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240		8b. OFFICE SYMBOL (If applicable) COES		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008	
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center		8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		10. SOURCE OF FUNDING NUMBERS	
				PROGRAM ELEMENT NO. 62702F	PROJECT NO. 5581
				TASK NO. 27	WORK UNIT ACCESSION NO. 13
11. TITLE (Include Security Classification) NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 Discussing, Using, and Recognizing Plans					
12. PERSONAL AUTHOR(S) Stuart C. Shapiro, Beverly Woolf					
13a. TYPE OF REPORT Interim		13b. TIME COVERED FROM Dec 86 TO Dec 87		14. DATE OF REPORT (Year, Month, Day) March 1989	
				15. PAGE COUNT 232	
16. SUPPLEMENTARY NOTATION This effort was performed as a subcontract by the State University of New York - Buffalo and the University of Massachusetts at Amherst to Syracuse University, Office of (Cont'd)					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Expert Systems, Semantic Networks, Planning, Natural Language Processing, Parsing. (JD)		
12	05				
12	07				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the third year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photo interpreta- tion, time-oriented problem solving, speech understanding systems, knowledge base mainten- ance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system. The specific topic for this volume is the development of a natural language interacting system with the ability to discuss, use, and recognize plans. This project requires the investigation of a knowledge representation design compatible with the intensional knowledge representation theory (Cont'd on reverse)					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL ROBERT L. RUSSEL, JR. Capt, USAF			22b. TELEPHONE (Include Area Code) (315) 330-3221		22c. OFFICE SYMBOL RADC/IRDP

DD Form 1473, JUN 86

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

Block 16 (Cont'd)
sponsored programs.

Block 19 (Cont'd)

previously developed by Dr. Stuart Shapiro and his co-workers at SUNY Buffalo (UB). In addition, the project involves an effort by Dr. Beverly Woolf and her co-workers at the University of Massachusetts to use the planning and plan recognition capabilities of this new system to better understand narrative text. The project officially got underway on August 13, 1987. To date, it has involved reviewing the extensive literature on planning, working on the software to be used jointly by the UB and U Mass. researchers, developing and experimenting with an initial design of a representation of plans, and investigating the possible integration of GRAPPLE, the U Mass plan formalism with SNePS, the UB knowledge representation system.

see Keywords on pg. R

UNCLASSIFIED

2B DISCUSSING, USING AND RECOGNIZING PLANS

Report submitted by:

Stuart C. Shapiro, Principal Investigator
Beverly Woolf, Co-PI, U.Mass.
Deepak Kumar, Graduate Research Assistant
Syed Ali, Graduate Research Assistant

Department of Computer Science
SUNY at Buffalo
226 Bell Hall
Buffalo, NY 14260



Approved for	
Name	<input checked="" type="checkbox"/>
Project	<input checked="" type="checkbox"/>
Unit	<input type="checkbox"/>
Justification	<input type="checkbox"/>
By	
Director	
Available to	
Dist	
A-1	

Table of Contents

2B.1 OVERVIEW	2B-5
2B.2 OBJECTIVES	2B-7
2B.3 TECHNICAL PROGRESS IN 1987	2B-9
2B.3.1 The Underlying Knowledge Representation Theory	2B-9
2B.3.2 A First Representation of Acts and Plans	2B-9
2B.3.3 An Acting System	2B-12
2B.3.4 The Blocks World	2B-13
2B.3.5 Conclusions	2B-22
2B.4 SOFTWARE DEVELOPMENT	2B-23
2B.5 REFERENCES	2B-25
2B.6 List of Publications - Discussing, Using and Recognizing Plans	2B-27
2B.7 Trips Funded by RADDC	2B-29
2B.8 Conferences Attended, Papers Presented, Sessions Chaired	2B-31
2B.9 Other Non-funded Activities and Papers by NAIC Supported Researchers	2B-33

2B.10 Selected Department AI Activities in 1987	2B-35
2B.10.1 New AI Faculty	2B-35
2B.10.2 AI Faculty	2B-35
2B.10.3 AI Seminars in 1987	2B-35
2B.10.4 Advanced Degrees Conferred in AI	2B-35

APPENDICES

Appendix B1: On Knowledge Representation Using Semantic Networks and Sanskrit	2B-36
Appendix B2: Knowledge-Based Parsing	2B-52
Appendix B3: Design of an Incremental Compiler for a Production- System ATN Machine	2B-97
Appendix B4: SNePS Considered as a Fully Intensional Propositional Semantic Network	2B-138

2B DISCUSSING, USING AND RECOGNIZING PLANS

2B.1 OVERVIEW

This project, also known as the Natural Language Planning project, is a joint project of a research group at SUNY at Buffalo (UB), led by Dr. Stuart C. Shapiro, and a research group at U. Mass. led by Dr. Beverly Woolf. The project is devoted to the investigation of a knowledge representation design compatible with the intensional knowledge representation theory previously developed by Dr. Shapiro and his co-workers and capable of providing a natural language interacting system with the ability to discuss, use, and recognize plans. The project officially got underway on August 13, 1987. These last few months of 1987 were devoted to reviewing the extensive literature on planning, working on the software to be used jointly by the UB and U. Mass. researchers, developing and experimenting with an initial design of a representation of plans, and investigating the possible integration of GRAPPLE, the U. Mass. plan formalism with SNePS, the UB knowledge representation system.

2B.2 OBJECTIVES

The objectives of this project are to:

- (1) design a representation for plans and rules for reasoning about plans within an established knowledge representation/reasoning (KRR) system; enhance the KRR system so that it can act according to such plans;
- (2) write a grammar to direct an established natural language processing (NLP) system to analyze English sentences about plans and represent the semantic/conceptual content of the sentences in the representation designed for objective (1); the resulting NLP system should be able to: accept sentences describing plans, and add the plans to its "plan library"; answer questions about the plans in its plan library; accept sentences describing the actions of others, and recognize when those actions constitute the carrying out of a plan in its plan library.

The KRR system to be used is SNePS (Shapiro 1979), and the NLP system to be modified for this purpose is CASSIE (Shapiro & Rapaport 1987). The UB group is responsible for enhancing SNePS/CASSIE according to the objectives listed above. The U. Mass. group is responsible for testing the enhanced system in the specific domains of the Blocks World, tutoring, and space launch narratives. Communication and feedback between the two groups will greatly improve the work of both.

2B.3 TECHNICAL PROGRESS IN 1987

2B.3.1 The Underlying Knowledge Representation Theory

A basic principle of our theory of intensional knowledge representation, embodied in the design and use of SNePS as a propositional semantic network, is The Uniqueness Principle – that there be a one-to-one mapping between nodes of the semantic network and concepts (mental objects) about which information may be stored in the network. These concepts are not limited to objects in the real world, but may be various ways of thinking about a single real world object (e.g. the Morning Star *vs.* the Evening Star *vs.* Venus). They may also be abstract objects like properties, propositions, Truth, Beauty, fictional objects, and impossible objects. They may include specific facts as well as general facts, and even rules, which can be believed, disbelieved, or followed when reasoning.

It is a major hypothesis of the current project that plans are also mental objects that can be represented in such a propositional semantic network. We can discuss plans with each other, reason about them, formulate them, follow them, and recognize when others seem to be following them. An AI system, using SNePS as its belief structure, should also be able to do these things.

Plans, being structures of actions, states, and other plans, resemble reasoning rules, which are structures of beliefs. However, they are different in important ways: reasoning rules are rules for forming new beliefs based on old beliefs, plans are rules for acting; a belief, once formed, need not be formed again, an action may need to be performed multiple times; the temporal order of assessing old beliefs and forming new beliefs is very flexible, the temporal order of performing actions is crucial to the correct carrying out of a plan. The representation of reasoning rules in SNePS, and the algorithm for reasoning according to them, implemented in SNIP – the SNePS Inference Package, have been carefully designed to make reasoning flexible and efficient. In this project, we are undertaking a similar design of the representation and use of plans. The various objectives, outlined in Section 2B.2, form important constraints on this design.

2B.3.2 A First Representation of Acts and Plans

In 1987, we have begun experimenting with an initial design of plans and of an acting system that uses them. Our representation of acts is based on that of Almeida (1987). He distinguishes the nodes for an act, the event of that act's being performed by a particular actor at a particular time, and the proposition of that event's having occurred. The benefit of this distinction is that the same node may be used to represent the same act (as required by the Uniqueness Principle) no matter who performs it and no matter when performed. Our initial representation of an act is a node with an ACTION arc to a node that represents the action, and OBJECT1, ..., OBJECTn arcs to the required objects of the action, as shown in Fig. 1. For example, the SNePSUL command for building a node representing the act of saying "FOO" is:

(build action say object1 FOO)

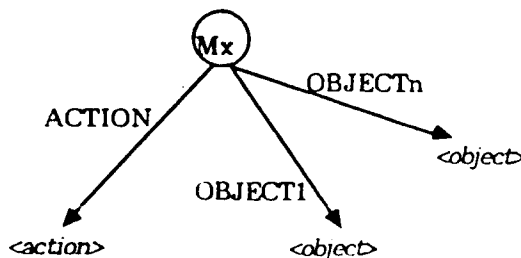


Figure 1. Mx represents the act of performing <action> on the <object>s.

It is necessary to distinguish between primitive and complex (non-primitive) actions. A primitive action is one that the system is capable of performing. The system is not capable of performing a complex action, so in order to carry out a complex action, the system must decompose it into a structure of other actions, which, ultimately, must be primitive. We assert that an action is primitive using the standard SNePS MEMBER-CLASS case frame shown in Fig. 2. To assert that "saying" is primitive, we execute the SNePSUL command:

```
(build member say class primitive)
```

Complex actions are those actions not asserted to be primitive.

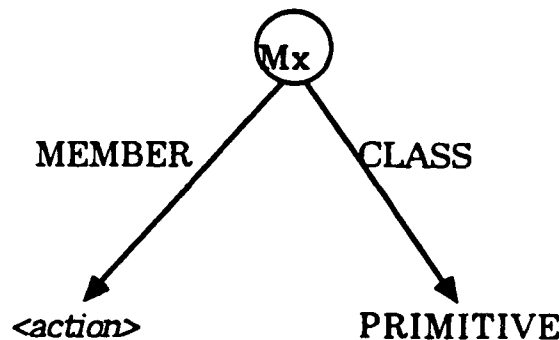


Figure 2. Mx represents the proposition that <action> is a primitive action.

A complex action is performed by decomposing it into a structure of simpler actions, which constitute a plan for carrying out the complex action. Our representation presupposes the hypothesis that this decomposition must take the objects of the action into account, but needn't take the actor into account. I.e., the decomposition relation holds between two acts, and is represented as shown in Fig. 3. The act at the end of the PLAN arc must be more decomposed than the act at the end of the ACT arc. Eg., it may be the case that act1 is complex, while act2 is primitive. (We will say that an act is primitive or complex just when its action is.) We can use a SNePS rule to assert that the plan for "asserting" anything is to "say" it by executing:

```
(build avb $x
  act (build action assert object1 *x)
  plan (build action say object1 *x))
```

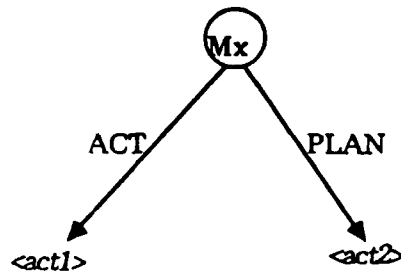


Figure 3. Mx represents the proposition that <act2> constitutes a plan for carrying out <act1>. <act2> must be more decomposed than <act1>.

This kind of plan is one for carrying out a complex action. Another kind of plan is one for achieving some state. The representation of that kind of plan is shown in Fig. 4. For the representations of propositions, we use the constructs shown in this report and those shown in (Shapiro & Rapaport 1987).

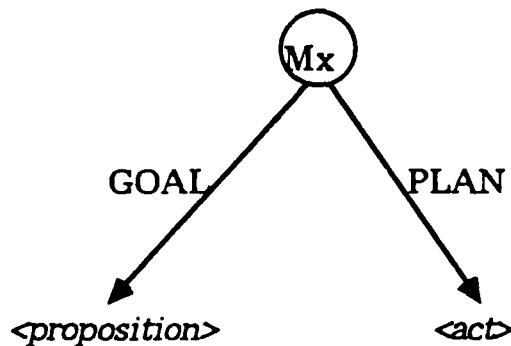


Figure 4. Mx represents the proposition that <act> constitutes a plan for achieving a state in which <proposition> is true.

The system may already know a plan for a goal or complex act. However, if it does not already have such a plan, it may try to produce it by reasoning about the complex act or goal, effects of acts, etc. Such reasoning constitutes the planning activity. Effects of an act may be asserted into the SNePS network just like any other beliefs. Initially, we use the representation shown in Fig. 5, although it is certainly true that the effects of an act may depend on the actor, so this representation is too simplistic. Another simplistic assumption we are making initially is that all effects of a performed act occur. Examples of effect assertions will be shown in the next section.

Initially, we are treating preconditions as conditions on plans. Examples will be shown in the next section. This is clearly inadequate, presupposing that planning is done "on the fly" as the acts are performed, and in the upcoming year, we will investigate advanced planning.

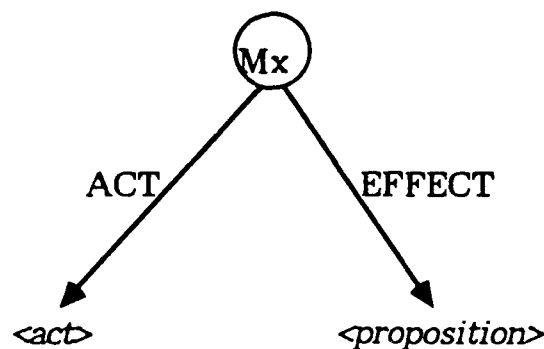


Figure 5. Mx represents the proposition that the effect of performing the <act> is that the <proposition> becomes true.

the use of states as preconditions, and reasoning about effects as a method of planning.

2B.3.3 An Acting System

The initial experimental acting system is composed of an acting executive (called "snact") and a queue of acts to be carried out. The top-level algorithm is:

```
SNACT(Queue) ::=
  REPEAT
    remove FIRST-ACT from Queue
    IF FIRST-ACT is primitive THEN
      DO FIRST-ACT
      INFER effects of FIRST-ACT,
        and schedule the believing of them
    ELSE { FIRST-ACT is complex }
      DEDUCE a plan for carrying out FIRST-ACT,
        and add it to Queue
    ENDIF
  UNTIL Queue is empty.
```

Primitive actions may be supplied to the system by programming them in Lisp. One of the simplest primitive action we have written is "say" -- the action of printing something on the output device, used as an example above. The code defining say is:

```
(defun say (n)
  (format t "~&~A~%"
    (first-atom (pathget n 'object1))))
```

Several things may be noted about this definition: the action takes an act node with itself as action as its argument so the function call of acts is consistent regardless of the number of arguments the action takes; pathget takes a node and a path of arc label, and returns the list of nodes at the end of the given path from the given node; first-atom returns the first node in the list under the assumption that each argument of a say act will be a single node. In the remainder of this report, we will refrain from showing actual Lisp code of primitive acts, but will describe them in the format:

```
(say <node>) Primitive-action
  "Prints <node> on the output device"
```

The arguments will be shown in a way that indicates their types.

So that we can explain the circumstances under which nothing need be done, we have defined the noop action:

```
(noop) Primitive-action
  "Does nothing."
```

The snact algorithm includes the scheduling of believing that the effects of acts have happened. This makes use of the primitive believe action:

(believe <proposition>) Primitive-action

"Causes the system to believe <proposition>, and removes from the system any belief in <proposition>'s negation."

Notice that this is an internal, mental action.

Another primitive action we have written is an important paradigm for our design and experimentation with representations of plans. It is the primitive action of performing two sub-acts in sequence:

(snsequence <act1> <act2>) Primitive-action

"Puts <act1> followed by <act2> on the front of the act queue"

Since either or both sub-acts can themselves be snsequence acts, we have a general structure for plans of sequential actions. Using this action as a model, we next will experiment with the design of other control structure actions.

Notice that snsequence is also a mental action, and is an "intention forming" action in the sense that putting an act on the act queue models a cognitive agent's forming the intention of performing that act.

One may also intend to achieve a goal. The goal, itself, being propositional, should not be put on the act queue. For that purpose, we have the achieve action:

(achieve <proposition>) Primitive-action

"Finds or infers a plan for achieving a state in which the proposition holds, and schedules the carrying out of that plan."

Since more than one plan may be found, we have defined a choose-plan operation. Initially, choose-plan chooses a plan at random, but we will investigate the creation of rules and plans for choosing plans more intelligently.

2B.3.4 The Blocks World

The first domain for our acting system is the well-known blocks world.

The predicates used in the Blocks World are:

(clear <block>) Predicate

"<block> has no blocks on top of it."

(ontable <block>) Predicate

"<block> is sitting directly on the table."

(holding <block>) Predicate

"The robot is holding <block>."

The Blocks World relations are:

(on <block1> <block2>) Relation

"<block1> is directly on top of <block2>."

The primitive Blocks World actions are:

```

(pickup <block>) Primitive-action
  "The robot picks up <block> from the table."

(putdown <block>) Primitive-action
  "The robot puts <block> down on the table."

(stack <block1> <block2>) Primitive-action
  "The robot stacks <block1> on top of <block2>."

(unstack <block1> <block2>) Primitive-action
  "The robot picks <block1> up from on top of <block2>."

```

Since we do not have an actual robot arm, these actions are simulated by printing appropriate messages to the output device.

The effects of the primitive actions are asserted by the following SNePSUL commands, where the preceding comments explain the statements.

```

;; After picking up a block, the block is not clear.
(build avb $block
  act (build action pickup object1 *block) = PICKUP-BLOCK
  effect (build min 0 max 0
    arg (build property clear object *block) = BLOCK-IS-CLEAR)
    = BLOCK-IS-NOT-CLEAR)

;; After picking up a block, the block is not on the table.
(build avb *block
  act *PICKUP-BLOCK
  effect (build min 0 max 0
    arg (build property ontable object *block) = BLOCK-ON-TABLE))

;; After picking up a block, the robot is holding the block.
(build avb *block
  act *PICKUP-BLOCK
  effect (build property holding object *block) = HOLDING-BLOCK)

;; After putting down a block, the robot is not holding the block.
(build avb *block
  act (build action putdown object1 *block) = PUTDOWN-BLOCK
  effect (build min 0 max 0 arg *HOLDING-BLOCK) = NOT-HOLDING-BLOCK)

;; After putting down a block, the block is clear.
(build avb *block
  act *PUTDOWN-BLOCK
  effect *BLOCK-IS-CLEAR)

;; After putting down a block, the block is on the table.
(build avb *block
  act *PUTDOWN-BLOCK
  effect *BLOCK-ON-TABLE)

;; After stacking one block on another, the robot is no longer holding it.
(build avb (*block $other-block)

```

```

act (build action stack object1 *block object2 *other-block)
  = STACK-ONE-ON-OTHER
effect *NOT-HOLDING-BLOCK)

```

≡ After stacking one block on another, the latter is no longer clear.

```

(build avb (*block *other-block)
  act *STACK-ONE-ON-OTHER
  effect (build min 0 max 0
    arg (build property clear object *other-block)
    = OTHER-IS-CLEAR))

```

≡ After stacking one block on another, the former is on the latter.

```

(build avb (*block *other-block)
  act *STACK-ONE-ON-OTHER
  effect (build rel on arg1 *block arg2 *other-block) = ONE-ON-OTHER)

```

≡ After stacking one block on another, the former is clear.

```

(build avb (*block *other-block)
  act *STACK-ONE-ON-OTHER
  effect *BLOCK-IS-CLEAR)

```

≡ After unstacking one block from another, it is no longer clear.

```

(build avb (*block *other-block)
  act (build action unstack object1 *block object2 *other-block)
    = UNSTACK-ONE-FROM-OTHER
  effect *BLOCK-IS-NOT-CLEAR)

```

≡ After unstacking one block from another, it is no longer on the other.

```

(build avb (*block *other-block)
  act *UNSTACK-ONE-FROM-OTHER
  effect (build min 0 max 0 arg *ONE-ON-OTHER))

```

≡ After unstacking one block from another, the latter is clear.

```

(build avb (*block *other-block)
  act *UNSTACK-ONE-FROM-OTHER
  effect *OTHER-IS-CLEAR)

```

≡ After unstacking one block from another, the robot is holding the former.

```

(build avb (*block *other-block)
  act *UNSTACK-ONE-FROM-OTHER
  effect *HOLDING-BLOCK)

```

Next, we describe some Blocks World plans.

Plans for achieving the holding of a block

≡ If the robot is not already holding some block, B1, and B1 is on some other block, B2, and B1 is clear, then unstacking B1 from B2 is a plan for holding B1.

```

(build avb (*block *other-block)
  &ant ((build I-/- t arg *HOLDING-BLOCK)
    *ONE-ON-OTHER
    *BLOCK-IS-CLEAR)

```

```

cq (build plan *UNSTACK-ONE-FROM-OTHER
    goal *HOLDING-BLOCK))

```

≡ If a block is on the table and clear, then picking it up is a plan for holding it.

```

(build avb *block
    &ant (*BLOCK-ON-TABLE
        *BLOCK-IS-CLEAR)
    cq (build plan *PICKUP-BLOCK
        goal *HOLDING-BLOCK))

```

≡ If the robot is already holding a block, then doing nothing is a plan for holding it.

```

(build avb *block
    ant *HOLDING-BLOCK
    cq (build plan (build action noop) = DO-NOTHING
        goal *HOLDING-BLOCK))

```

Plans for getting a block on the table

≡ If some block is not on the table, then the robot can get it there by first holding the block, then putting it down.

```

(build avb *block
    ant (build I/- t arg *BLOCK-ON-TABLE)
    cq (build plan (build action snsequence
        object1 (build action achieve
            object1 *HOLDING-BLOCK)
            = HOLD-BLOCK
            object2 *PUTDOWN-BLOCK)
        goal *BLOCK-ON-TABLE))

```

≡ If a block is already on the table, then doing nothing is a plan for putting it there.

```

(build avb *block
    ant *BLOCK-ON-TABLE
    cq (build plan *DO-NOTHING
        goal *BLOCK-ON-TABLE))

```

Plans for getting one block on top of another

≡ If block B1 is already on top of block B2, then one need do nothing to get it there.

```

(build avb (*block *other-block)
    ant *ONE-ON-OTHER
    cq (build plan *DO-NOTHING
        goal *ONE-ON-OTHER))

```

≡ If block B1 is not already on top of block B2, then to put it there, first clear B2, then hold B1, then stack B1 on top of B2.

```

(build avb (*block *other-block)
    ant (build I/- t arg *ONE-ON-OTHER)
    cq (build plan (build action snsequence
        object1 (build action achieve

```

```

                                object1 *OTHER-IS-CLEAR)
                                object2 (build action snsequence
                                object1 *HOLD-BLOCK
                                object2 *STACK-ONE-ON-OTHER))
goal *ONE-ON-OTHER))

```

Plans for clearing a block

;; If a block is already clear, nothing needs to be done to clear it.

```

(build avb *block
  ant *BLOCK-IS-CLEAR
  cq (build plan *DO-NOTHING
    goal *BLOCK-IS-CLEAR))

```

;; If block B1 is on top of block B2, then to clear B2,

;; first clear B1, then put B1 on the table.

```

(build avb (*block *other-block)
  ant *ONE-ON-OTHER
  cq (build plan (build action snsequence
    object1 (build action achieve
      object1 *BLOCK-IS-CLEAR)
    object2 (build action achieve
      object1 *BLOCK-ON-TABLE))
    goal *OTHER-IS-CLEAR))

```

A plan for building a stack of three blocks

;; To build a stack of three blocks, B1 on B2 on B3,

;; first put B3 on the table, then put B2 on B3, then put B1 on B2.

```

(build avb (*block *other-block $third-block)
  act (build action make-3-stack
    object1 *third-block
    object2 *block
    object3 *other-block)
  plan (build action snsequence
    object1 (build action achieve
      object1 (build property ontable
        object *other-block))
    object2 (build action snsequence
      object1
        (build action achieve
          object1 *ONE-ON-OTHER)
      object2
        (build action achieve
          object1 (build rel on
            arg1 *third-block
            arg2 *block))))))

```

The following is a run of the system after loading the actions, effects, plans, etc. discussed above. Comments are on lines beginning with ";". The SNePS prompt is ";;". Other lines are SNePS output. The only editing has been the elimination of extra blank lines and some stray trace prints.

First, we will ask the system some questions about the actions and plans it knows about.

```
; What actions do you know about?  
* (find action- ?x)
```

```
(make-3-stack unstack stack putdown pickup noop achieve snsequence)  
exec: 0.26 sec   gc: 0.00 sec
```

```
; What are the primitive actions?  
* (find (member- class) primitive)
```

```
(unstack stack putdown pickup forget believe noop say achieve snsequence)  
exec: 0.06 sec   gc: 0.00 sec
```

```
; What complex actions do you know about?  
* ((find action- ?x) - (find (member- class) primitive))
```

```
(make-3-stack)  
exec: 0.26 sec   gc: 0.00 sec
```

```
; What are the effects of picking up a block?  
* (desc (find (act action) pickup effect ?x))
```

```
(m16 (effect (m15 (object (v1)) (property (holding))))  
      (act (m8 (object1 (v1)) (action (pickup))))  
      (avb (v1))))  
(m14 (effect  
      (m13 (arg (m12 (object (v1)) (property (ontable)))) (max (0)) (min (0))))  
      (act (m8 (object1 (v1)) (action (pickup))))  
      (avb (v1))))  
(m11 (effect  
      (m10 (arg (m9 (object (v1)) (property (clear)))) (max (0)) (min (0))))  
      (act (m8 (object1 (v1)) (action (pickup))))  
      (avb (v1))))  
(dumped)  
exec: 0.31 sec   gc: 0.00 sec
```

```
; How would you make a stack of three blocks?  
* (desc (find (act action) make-3-stack plan ?x))
```

```
(m74 (plan  
      (m73 (object2  
            (m72 (object2  
                  (m71 (object1 (m70 (arg2 (v1)) (arg1 (v3)) (rel (on))))  
                    (action (achieve))))  
            (object1  
              (m69 (object1 (m27 (arg2 (v2)) (arg1 (v1)) (rel (on))))  
                (action (achieve))))  
            (action (snsequence))))  
      (object1  
        (m68 (object1 (m67 (object (v2)) (property (ontable))))  
          (action (achieve))))  
      (action (snsequence))))
```

```

(act
  (m66 (object3 (v2))
    (object2 (v1))
    (object1 (v3))
    (action (make-3-stack))))
  (avb (v3) (v2) (v1)))
(dumped)
exec: 0.28 sec   gc: 0.00 sec

```

We will now demonstrate having this system solve the Blocks World problem shown in Fig. 6.

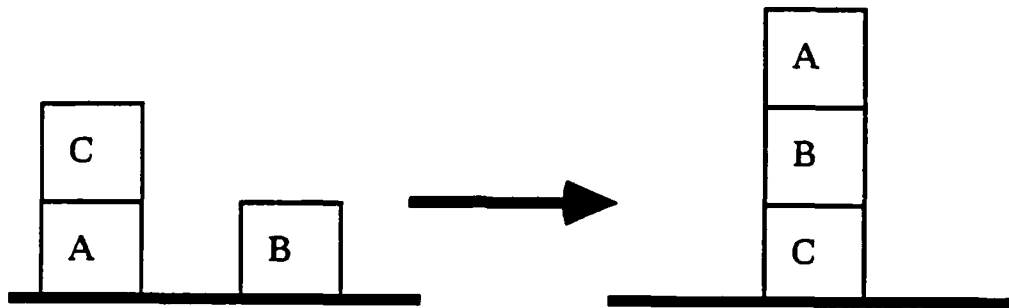


Figure 6. A Blocks World problem to be solved.

≡ First, we define the initial state of the problem.

```

; C is clear.
* (describe (forbtop property clear object C))
(m75 (object (C)) (property (clear)))
(dumped)
exec: 0.15 sec   gc: 0.00 sec

```

```

; A is on C.
* (describe (forbtop rel on arg1 C arg2 A))
(m76 (arg2 (A)) (arg1 (C)) (rel (on)))
(dumped)
exec: 0.10 sec   gc: 0.00 sec

```

```

; A is on the table.
* (describe (forbtop property ontable object A))
(m77 (object (A)) (property (ontable)))
(dumped)
exec: 0.08 sec   gc: 0.00 sec

```

```

; B is clear.
* (describe (forbtop property clear object B))
(m78 (object (B)) (property (clear)))

```

(dumped)

exec: 0.10 sec gc: 0.00 sec

; B is on the table.

* (describe (forbtop property ontable object B))

(m79 (object (B)) (property (ontable)))

(dumped)

exec: 0.10 sec gc: 0.00 sec

;; Then we ask the system to build a stack.

; Make a stack of A on B on C.

* (snact (build action make-3-stack object1 A object2 B object3 C))

Now doing: UNSTACK C from A.

Now doing: DISBELIEVE:

(m75 (object (C)) (property (clear)))

Now doing: DISBELIEVE:

(m76 (arg2 (A)) (arg1 (C)) (rel (on)))

Now doing: BELIEVE:

(m104 (object (A)) (property (clear)))

Now doing: BELIEVE:

(m92 (object (C)) (property (holding)))

Now doing: PUTDOWN C on table.

Now doing: DISBELIEVE:

(m92 (object (C)) (property (holding)))

Now doing: BELIEVE:

(m75 (object (C)) (property (clear)))

Now doing: BELIEVE:

(m87 (object (C)) (property (ontable)))

Now doing: NOOP

Now doing: PICKUP B from table.

Now doing: DISBELIEVE:

(m78 (object (B)) (property (clear)))

Now doing: DISBELIEVE:

(m79 (object (B)) (property (ontable)))

Now doing: BELIEVE:

(m126 (object (B)) (property (holding)))

Now doing: STACK B on C.

Now doing: DISBELIEVE:
(m126 (object (B)) (property (holding)))

Now doing: DISBELIEVE:
(m75 (object (C)) (property (clear)))

Now doing: BELIEVE:
(m84 (arg2 (C)) (arg1 (B)) (rel (on)))

Now doing: BELIEVE:
(m78 (object (B)) (property (clear)))

Now doing: NOOP

Now doing: PICKUP A from table.

Now doing: DISBELIEVE:
(m104 (object (A)) (property (clear)))

Now doing: DISBELIEVE:
(m77 (object (A)) (property (ontable)))

Now doing: BELIEVE:
(m163 (object (A)) (property (holding)))

Now doing: STACK A on B.

Now doing: DISBELIEVE:
(m163 (object (A)) (property (holding)))

Now doing: DISBELIEVE:
(m78 (object (B)) (property (clear)))

Now doing: BELIEVE:
(m82 (arg2 (B)) (arg1 (A)) (rel (on)))

Now doing: BELIEVE:
(m104 (object (A)) (property (clear)))

nil

exec: 36.00 sec gc: 7.60 sec

We can now find out the final state by asking about all the predicates and relations.

; What objects are clear?
*(describe (deduce property clear object %x))

(m104 (object (A)) (property (clear)))
(m150 (arg (m75 (object (C)) (property (clear)))) (max (0)) (min (0)))
(m186 (arg (m78 (object (B)) (property (clear)))) (max (0)) (min (0)))
(dumped)

exec: 1.83 sec gc: 0.00 sec

```

; What objects are on the table?
*(describe (deduce property ontable object %x))

(m87 (object (C)) (property (ontable)))
(m175 (arg (m77 (object (A)) (property (ontable)))) (max (0)) (min (0)))
(m139 (arg (m79 (object (B)) (property (ontable)))) (max (0)) (min (0)))
(dumped)
exec: 1.71 sec   gc: 0.00 sec

; What objects are being held?
*(describe (deduce property holding object %x))

(m115 (arg (m92 (object (C)) (property (holding)))) (max (0)) (min (0)))
(m148 (arg (m126 (object (B)) (property (holding)))) (max (0)) (min (0)))
(m184 (arg (m163 (object (A)) (property (holding)))) (max (0)) (min (0)))
(dumped)
exec: 2.35 sec   gc: 0.00 sec

; What objects are on what other objects?
*(describe (deduce rel on arg1 %x arg2 %y))

(m82 (arg2 (B)) (arg1 (A)) (rel (on)))
(m84 (arg2 (C)) (arg1 (B)) (rel (on)))
(m102 (arg (m76 (arg2 (A)) (arg1 (C)) (rel (on)))) (max (0)) (min (0)))
(dumped)
exec: 1.35 sec   gc: 0.00 sec

```

2B.3.5 Conclusions

We have designed and implemented a first version of representing acts and plans in SNePs, and of a SNePS acting component. We have shown primitive acts, their effects, plans for complex acts, and plans for achieving goals being explained to the system in short declarations, each of which can be seen to be expressible in an easily understood English sentence. We have shown the system answering questions about its actions and plans. We have shown the system using its plans to carry out a Blocks World problem.

Our representation distinguishes actions, acts, propositions, decomposition (act-based) plans, and goal-oriented (state-based) plans. It also distinguishes between primitive actions/acts and complex actions/acts. We have actions that the system performs on the world (simulated by printing onto the output device), such as pickup and putdown, and internal, mental actions, such as believe. We also have intention-forming actions, such as achieve, whose effects are to place acts on the act queue. The intention-forming action *snsequence* is a control-structure action which permits us to represent and carry out structured plans.

Our next tasks are clear. We will modify the current CASSIE grammar so that the kinds of conversations shown in this report can be carried out in English instead of in SNePSUL. We will modify our Blocks World representation to more closely mimic that of (Huff & Lesser 1987). We will add additional primitive control structure actions. We will modify our representation of plans so that advanced planning can be carried out. We will begin investigating the use of these plans to recognize when other actors are carrying out plans the system knows about. We will begin representing plans from the tutoring domain.

2B.4 SOFTWARE DEVELOPMENT

SNePS-2 is a new design of SNePS, incorporating advances in our theory of intensional knowledge representation made since the original SNePS was designed and implemented. SNePS-2 is being implemented in Common Lisp on Texas Instruments Explorers, Symbolics Lisp Machines, and HP AI Workstations. The current implementation of SNePS-2 includes the core SNePS functions of explicit storage into and retrieval from SNePS networks, and a Generalized ATN (Shapiro 1982) Grammar interpreter. This implementation has been delivered by the group at UB to the group at U. Mass. Implementation of SNIP-2 has begun.

2B.5 REFERENCES

- (1) Almeida, M. J. Reasoning about the Temporal Structure of Narratives, Technical Report No. 87-10, Department of Computer Science, SUNY at Buffalo, Buffalo, NY, 1987.
- (2) Huff, K. E. & Lesser, V. R. The GRAPPLE Plan Formalism, COINS Technical Report 87-08, Department of Computer and Information Science, U. Mass., Amherst, MA, 1987.
- (3) Shapiro, S. C. The SNePS semantic network processing system. In N. V. Findler, ed. *Associative Networks: The Representation and Use of Knowledge by Computers*. Academic Press, New York, 1979, 179-203.
- (4) Shapiro, S. C. Generalized augmented transition network grammars for generation from semantic networks. *American Journal of Computational Linguistics* 8, 1 (January-March 1982), 12-25.
- (5) Shapiro, S. C. & Rapaport, W. J. SNePS considered as a fully intensional propositional semantic network. In G. McCalla & N. Cercone, eds. *The Knowledge Frontier: Essays in the Representation of Knowledge*. Springer-Verlag, New York, 1987, 262-315.

2B.6 LIST OF PUBLICATIONS - DISCUSSING, USING AND RECOGNIZING PLANS

- (1) Srihari, S. N., Rapaport, W. J., Kumar, D. [1987] "On Knowledge Representation Using Semantic Networks and Sanskrit", Technical Report 87-03, Department of Computer Science, SUNY at Buffalo, Buffalo, NY.
- (2) Neal, J. G. and Shapiro, S. C., [1987] Knowledge-Based Parsing, *Natural Language Parsing Systems*, L. Bolc, ed., Springer-Verlag, Berlin, 49-92.
- (3) Yuhan, A. H. and Shapiro, S. C. [1987] "Design of an Incremental Compiler for a Production-system ATN Machine", Technical Report No. 87-09, Department of Computer Science, SUNY at Buffalo, 38 pp.
- (4) S. C. Shapiro, R. J. Rapaport, [1987] SNePS Considered as a Fully Intensional Propositional Semantic Network, *The Knowledge Frontier*, N. Cercone, G. McCalla, eds., Springer-Verlag, New York, 263-315.

2B.7 TRIPS FUNDED BY RADC

Reliability and Maintainability Symposium, Philadelphia, PA, January 27-29, 1987: Shapiro.

NAIC Executive Committee Meeting, RPI, February 2, 1987: Shapiro.

Conference on AI Applications by IEEE, Orlando, Florida, February 25-27, 1987: Taie.

NAIC Executive Committee Meeting, Syracuse, NY, March 27, 1987: Shapiro.

NAIC Spring Technology Fair, RADC, April 9-10, 1987: Shapiro, Srihari, Geller, Taie, Campbell.

NAIC Summer Workshop, Minnowbrook Convention Center, NY, June 29 - July 2, 1987: Shapiro, Srihari, Geller, Bettinger.

Tenth International Joint Conference on Artificial Intelligence-87 (IJCAI-87), Milan, Italy, August 23-28, 1987: Shapiro, Geller.

SUNYAB VMES - NLP Meetings, Buffalo, NY, September 16-17, 1987: Shapiro.

Natural Language Planning Workshop, Minnowbrook Conference Center, NY, September 20-23, 1987: Shapiro.

NAIC Fall Conference, Potsdam, NY, October 2, 1987: Srihari, Kumar, Ali.

SNePS Workshop, U. Mass., December 1, 1987: Shapiro.

NAIC Executive Committee Meeting, SUNY at Buffalo, NY, December 3, 1987: Srihari, Spahr.

SUNYAB VMES - NLP Meetings, Buffalo, NY, December 2-4, 1987: Shapiro.

Concurrent Common Lisp Workshop, Beaverton, Oregon, December 9-11, 1987: Campbell.

2B.8 CONFERENCES ATTENDED, PAPERS PRESENTED

Conference on AI Applications by IEEE

Orlando, Florida, February 25-27, 1987

presented M. R. Taie Modeling Connections for Circuit Diagnosis

1987 Reliability and Maintainability Symposium

Philadelphia, PA, January 27-29, 1987,

presented S. C. Shapiro Knowledge Based Modeling of Circuit Boards

Second UB Graduate-Conference on Computer Science

Buffalo, NY, March 10, 1987

chairmen S. S. Campbell

editors J. Geller

K. E. Bettinger

presented T. F. Pawlicki

J. M. Mellor-Crummey The Representation of Visual Knowledge

S. Wroblewski, T. Thomas Parallel Program Debugging with Partial Orders
Efficient Trouble Shooting in an Industrial Environment

C.-H. Wang ABLS: An Object Recognition System for Locating Address Blocks on Mail Pieces

D. Horton, G. Hirst Presuppositions as Beliefs: A New Approach

N. Wahl, S. Miller Hypercube Algorithms to determine Geometric Properties of Digitized Pictures

D. Walters, G. Krishnan Bottom-up Analysis for Color Separation

B. Selman Vivid Representations and Analogues

S. Svorou The Spatial Extension Terms in Modern Greek

Y. Jang, H. K. Hung Semantics of a Recursive Procedure with Parameter and Aliasing

J. Tenenbergs, L. Hartman Naive Physics and the Control of Inference

Z. Xiang Multi-level Model-based Diagnosis Reasoning

Seventh International Workshop on Expert Systems

Avignon, France, May 13-15, 1987

presented S. N. Srihari Tutorial on Spatial Knowledge Representation and Reasoning

Sixth National Conference on AI

Seattle, WA, July 13-17, 1987

presented J. J. Hull

G. Krishnan Hypothesis Testing in a Computational Theory

of Visual Word Recognition

Adding Vision Capabilities to a Computer Graphics System for Color Separation

other attendees S. N. Srihari

D. Walters

D. Niyogi

K. Bettinger

D. Kumar

Z. Dobes

AAAI Workshop on Blackboard Architectures

Seattle, WA, July 15, 1987

presented C.-H. Wang

Object Reception in Structured and Random Environments Using a Blackboard Architecture

other attendees S. N. Srihari

International Joint Conference on Artificial Intelligence

Milan, Italy, August 22-24, 1987

presented J. J. Hull

Knowledge Utilization in Handwritten Zip Code Recognition

R. M. Bozinovic

A Multi-level Perception Approach to Reading Cursive Script

J. Geller

Graphical Deep Knowledge for Intelligent Machine Drafting

S. Peters

A Representation for Natural Catagory Systems

other attendees S. N. Srihari

S. C. Shapiro

SPSE's 40th Annual Conference and Symposium on Hybrid Image Systems

Rochester, NY, May 17-22, 1987

presented S. N. Srihari

Document Image Analysis: An Overview of Algorithms

J. J. Hull

A Computational Model for Human and Machine Reading

International Workshop on Expert Systems and Pattern Recognition

Novosibirsk, USSR, Oct. 25-28, 1987

presented S. N. Srihari

Spatial Knowledge Representation and Reasoning

2B.9 OTHER NON-FUNDED ACTIVITIES AND PAPERS BY NAIC SUPPORTED RESEARCHERS

The *Encyclopedia of Artificial Intelligence*, edited by Dr. S. C. Shapiro, was published by John Wiley & Sons, Inc., New York, 1987.

Dr. Srihari presented his paper (not sponsored by RADC), "Spatial Knowledge Representation and Reasoning", at the Seventh International Workshop Conference on Expert Systems and Their Applications, in Avignon, France, May 13-15, 1987. He is sponsored by ECCAI - European Coordinating Committee for Artificial Intelligence.

The Second Annual University at Buffalo Graduate-Conference on Computer Science was held on March 10, 1987. This is a one day conference organized and presented completely by SUNYAB Computer Science graduate students. Scott Campbell was the chairman; Keith Bettinger and James Geller were included on the committee. Twelve graduates from four departments in three universities presented their research work in C.S. - eight of which were AI related talks. In order to continue and expand on NAIC's principle of enhanced communication between member institutions, two of the presentors were students at the University of Rochester. We feel that this event has helped to strengthen lines of communication between our two Universities. The event was well attended with a total 146 attendees - a 21.6% increase over last year's attendance. Many of the attendees were from local businesses and universities, who saw how much our involvement in the NAIC has strengthened our department, as well as what is going on in the NAIC.

Dr. Shapiro went to Bolling AFB, Washington, D. C., April 3-4 (sponsored by the National Research Council, Board on Mathematical Sciences) to attend a meeting of the Review Panel for the Research Program of the Mathematical and Information Sciences Directorate, AFOSR.

Dr. Shapiro demonstrated one of the Explorers at the UB Freshman Open House, April 11.

Dr. Shapiro presented "CASSIE: Development of a Computational Mind", Department Colloquium, Department of Computer Science, SUNY at Albany, April 8, 1987.

Dr. Shapiro led a discussion on AI with a group of undergraduate students of Wilkeson Quadrangle, UB, as part of the FAST (Faculty and Students Talking) series, April 28.

Dr. Shapiro presented "Toward a Computational Mind", at a meeting of the Niagara Frontier Chapter of the ACM, Buffalo, NY, May 7.

Dr. Srihari (not sponsored by RADC) traveled to Avignon, France, May 9-15, 1987, and presented his paper, "Spatial Knowledge Representation and Reasoning", at the Seventh International Workshop Conference on Expert Systems and Their Applications. He was sponsored by ECCAI - European Coordinating Committee for Artificial Intelligence.

Keith Bettinger attended AAI-87, July 13-17, 1987, in Seattle, Washington as a volunteer.

Dr. Sargur N. Srihari (not funded by RADC) attended the AAI-87 Sixth National Conference on Artificial Intelligence at the University of Washington, Seattle WA, July 13-17, 1987. He participated in the Blackboard Systems Workshop.

Dr. Shapiro co-authored the paper (not funded by RADC) with Sandy Peters, "A Representation for Natural Category Systems", presented at the International Joint Conference on Artificial Intelligence-87 (IJCAI-87) in Milan, Italy, August 23-28, 1987.

Sargur N. Srihari and Radmilo Bozinovic presented a paper "A Multilevel Perception Approach to Cursive Script Recognition", at the Tenth International Joint Conference on Artificial Intelligence-87 (IJCAI-87) in Milan, Italy, August 23-28, 1987.

Jonathon Hull and Sargur N. Srihari presented a paper, "Use of External Information in Zip-code Recognition", also at the same conference, the Tenth International Joint Conference on Artificial Intelligence-87 (IJCAI-87) in Milan, Italy, August 23-28, 1987.

Sargur N. Srihari and Radmilo Bozinovic have published a paper in the *Artificial Intelligence* Journal, October 1987 issue. The title of the paper is the same as the IJCAI paper, "A Multilevel Perception Approach to Cursive Script Recognition".

Dr. Shapiro spent Sept. 16-17 at the Department at UB meeting with the VMES and Natural Language Planning research groups.

Dr. Shapiro attended the 1987 Natural Language Planning Workshop in the Minnowbrook Conference Center, Sept. 20-23.

Dr. Shapiro presented "Semantic Network Based Reasoning Systems" at ISI, Marina del Rey, CA, on Sept. 30, 1987.

Dr. Shapiro gave a talk "CASSIE: Development of a Computational Mind" at the Computer Science Seminar, Department of Computer Science, University of Southern California, Oct. 28, 1987.

The paper "A Model for Belief Revision" by Stuart C. Shapiro and Joao P. Martins has been accepted for publication in the journal *Artificial Intelligence*, and is tentatively scheduled to appear in Winter, 1988.

Dr. Sargur N. Srihari (not funded by NAIC) attended the International Workshop on Expert Systems and Pattern Recognition, USSR Academy of Science (Siberian Section), Novosibirsk, USSR, October 26-30, 1987. He presented a paper on Spatial Knowledge Representation.

A paper written by Srihari, Wang, Palumbo, and Hull has been published in the Dec. 18, 1987 issue of *AI Magazine*. The article is featured as the lead article of the issue on the cover. *AI Magazine* is the principal publication of the American Association of Artificial Intelligence.

Professor Stuart C. Shapiro has been appointed to the program committee of the First International Conference on Principles of Knowledge Representation and Reasoning, to be held in Toronto, Canada, May 15-18, 1989.

2B.10 SELECTED DEPARTMENT ACTIVITIES IN 1987

2B.10.1 New AI Faculty

David Sher, Ph.D., accepted an appointment as Assistant Professor starting in Fall, 1987. Dr. Sher's research, in Computer Vision, was part of Dr. Chris Brown's task within the NAIC.

2B.10.2 AI Faculty

AI Faculty

Six of the fourteen faculty at SUNY at Buffalo are AI. They include: Shoshana Hardt; William J. Rapaport; David B. Sher; Stuart C. Shapiro; Sargur N. Srihari; Deborah D.K. Walters.

2B.10.3 AI Seminars in 1987

CS702 - Walters

Detection and Representation of Visual Features - Spring 87

CS703 - Hardt

Automating Intelligent Interaction with Complex Worlds - Spring 1987

CS705 - Leyton

Geometry of Robot Planning

CS706 - Srihari

Introduction to Connectionist/Neural Network Models

2B.10.4 Advanced Degrees Conferred in AI

AI M.S. degrees: Brinkerhoff L.; Campbell, S. C.; Chan, C. M.; Chen, Y. Y.; Chun, S. A.; DeVinney, G.; Dodson-Simmons, O.; Feuerstein, S.; Gupta, R.; Jian, H.; Kim, J.; Krishnaswamy, V.; Kuo, C. K.; Lang, S. L.; Lee, H. C.; Li, N.; Li, P.; Lively, R.; Murty, K.; Schwartz, M.; Siracusa, T.; Schneck, N. T.; So, H. M.; Thomas, T.; Wang, G.; Wroblewski, S.; Wu, T. Y.; Wu, W. J.

AI Ph.D.s:

George L. Sicherman

Thesis Supervisor: Shoshana L. Hardt

"A Model of Probabilistic Inference for Decision-Making Under Risk and Uncertainty"

Mingruey R. Taie

Thesis Supervisor: Sargur N. Srihari

"Representation of Device Knowledge for Versatile Fault Diagnosis"

Michael Almeida

Thesis Supervisor: Stuart C. Shapiro

"Reasoning about the Temporal Structure of Narratives"

Appendix B1:

"On Knowledge Representation Using Semantic Networks and Sanskrit"

Srihari, S.N.
Rapaport, W.J.
Kumar, D.

**ON KNOWLEDGE REPRESENTATION USING
SEMANTIC NETWORKS AND SANSKRIT**

S.N. Srihari, W.J. Rapaport, D. Kumar

87-03

February 1987

**Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, New York 14260**

This work was supported in part by the National Science Foundation under grant no. IST-8504713, and in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, NY 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract no. F30602-85-C-0008.

1. INTRODUCTION

Computational linguistics is a subfield of artificial intelligence (AI) concerned with the development of methodologies and algorithms for processing natural-language by computer. Methodologies for computational linguistics are largely based on linguistic theories, both traditional and modern. Recently, there has been a proposal to utilize a traditional method, viz., the shastric Sanskrit method of analysis (Briggs, 1985), as a knowledge representation formalism for natural-language processing. The proposal is based on the perceived similarity between a commonly used method of knowledge representation in AI, viz., semantic networks, and the shastric Sanskrit method, which is remarkably unambiguous.

The influence of Sanskrit on traditional Western linguistics is acknowledgedly significant (Gelb, 1985). While linguistic traditions such as Mesopotamian, Chinese, Arabic, etc., are largely enmeshed with their particularities, Sanskrit has had at least three major influences. First, the unraveling of Indo-European languages in comparative linguistics is attributed to the discovery of Sanskrit by Western linguists. Second, Sanskrit provides a phonetic analysis method which is vastly superior to Western phonetic tradition and its discovery led to the systematic study of Western phonetics. Third, and most important to the present paper, the rules of analysis (e.g., sutras of Panini) for compound nouns, etc., is very similar to contemporary theories such as those based on semantic networks.

The purpose of this paper is threefold: (i) to describe propositional semantic networks as used in AI, as well as a software system for semantic network processing known as SNePS (Shapiro, 1979), (ii) to describe several case structures that have been proposed for natural-language processing and which are necessary for natural language understanding based on semantic networks, and (iii) to introduce a proposal for natural-language translation based on shastric Sanskrit and semantic networks as an interlingua.

2. KNOWLEDGE REPRESENTATION USING SEMANTIC NETWORKS

2.1. Semantic Networks

A semantic network is a method of knowledge representation that has associated with it procedures for representing information, for retrieving information from it, and for performing inference with it. There are at least two sorts of semantic networks in the AI literature (see Findler 1979 for a survey): The most common is what is known as an "inheritance hierarchy," of which the most well-known is probably KL-ONE (cf. Brachman & Schmolze 1985). In an inheritance semantic network, nodes represent concepts, and arcs represent relations between them. For instance, a typical inheritance semantic network might represent the propositions that Socrates is human and that humans are mortal as in Figure 1(a). The interpreters for such systems allow properties to be "inherited," so that the fact that Socrates is mortal does not also have to be stored at the Socrates-node. What is essential, however, is that the representation of a proposition (e.g., that Socrates is human) consists only of separate representations of the individuals (Socrates and the property of being human) linked by a relation arc (the "ISA" arc). That is, propositions are not themselves objects.

[Figure 1 here]

In a *propositional* semantic network, all information, including propositions, is represented by nodes. The benefit of representing propositions by nodes is that propositions about propositions can be represented with no limit. Thus, for example, the information represented in the inheritance network of Figure 1(a) could (though it need not) be represented as in Figure 1(b); the crucial difference is that the propositional network contains nodes (m3, m5) representing the *propositions* that Socrates is human and that humans are mortal, thus enabling representations of beliefs and rules *about* those propositions.

2.2. SNePS

SNePS, the Semantic Network Processing System, is a knowledge-representation and reasoning software system based on propositional semantic networks. It has been used to model a cognitive agent's understanding of natural-language, in particular, English (Shapiro 1979; Maida & Shapiro 1982; Shapiro & Rapaport 1986, 1987; Rapaport 1986). SNePS is implemented in the LISP programming language and currently runs in Unix- and LISP-machine environments.

Arcs merely form the underlying syntactic structure of SNePS. This is embodied in the restriction that one cannot add an arc between two existing nodes. That would be tantamount to telling SNePS a proposition that is not represented as a node. Another restriction is the *Uniqueness Principle*: There is a one-to-one correspondence between nodes and represented concepts. This principle guarantees that nodes will be shared whenever possible and that nodes represent intensional objects. (Shapiro & Rapaport 1987.)

SNePS nodes that only have arcs pointing to them are considered to be unstructured or *atomic*. They include: (1) *sensory* nodes, which—when SNePS is being used to model a cognitive agent—represent interfaces with the external world (in the examples that follow, they represent utterances); (2) *base* nodes, which represent individual concepts and properties; and (3) *variable* nodes, which represent arbitrary individuals (Fine 1983) or arbitrary propositions.

Molecular nodes, which have arcs emanating from them, include: (1) *structured individual* nodes, which represent structured individual concepts or properties (i.e., concepts and properties represented in such a way that their internal structure is exhibited)—for an example, see Section 3, below; and (2) *structured proposition* nodes, which represent propositions; those with no incoming arcs represent *beliefs* of the system. (Note that structured proposition nodes can also be considered to be structured individuals.) Proposition nodes are either *atomic* (representing atomic propositions) or are *rule nodes*. Rule nodes represent deduction rules and are used for node-based deductive inference (Shapiro 1978; Shapiro & McKay 1980; McKay & Shapiro 1981; Shapiro, Martins, & McKay 1982). For each of the three categories of molecular nodes (structured individuals, atomic propositions, and rules), there are *constant* nodes of that category and *pattern* nodes of that category representing arbitrary entities of that category.

There are a few built-in arc labels, used mostly for rule nodes. *Paths* of arcs can be defined, allowing for *path-based* inference, including property inheritance within generalization hierarchies (Shapiro 1978, Srihari 1981). All other arc labels are defined by the user, typically at the beginning of an interaction with SNePS. In fact, since most arcs are user-defined, users are obligated to provide a formal syntax and semantics for their SNePS networks. We provide some examples, below.

Syntax and Semantics of SNePS

In this section, we give the syntax and semantics of the nodes and arcs used in the interaction. (A fuller presentation, together with the rest of the conversation, is in Shapiro & Rapaport 1986, 1987.)

(Def. 1) A node *dominates* another node if there is a path of directed arcs from the first node to the second node.

(Def. 2) A *pattern* node is a node that dominates a variable node.

(Def. 3) An *individual* node is either a base node, a variable node, or a structured constant or pattern individual node.

(Def. 4) A *proposition* node is either a structured proposition node or an atomic variable node representing an arbitrary proposition.

(Syn.1) If *w* is a(n English) word and *i* is an identifier not previously used, then



is a network, w is a sensory node, and i is a structured individual node.

(Sem.1) i is the object of thought corresponding to the utterance of w .

(Syn.2) If either t_1 and t_2 are identifiers not previously used, or t_1 is an identifier not previously used and t_2 is a temporal node, then



is a network and t_1 and t_2 are temporal nodes, i.e. individual nodes representing times.

(Sem.2) t_1 and t_2 are objects of thought corresponding to two times, the former occurring before the latter.

(Syn.3) If i and j are individual nodes and m is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.3) m is the object of thought corresponding to the proposition that i has the property j .

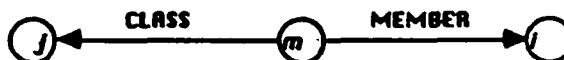
(Syn.4) If i and j are individual nodes and m is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.4) m is the object of thought corresponding to the proposition that i 's proper name is j . (j is the object of thought that is i 's proper name; its expression in English is represented by a node at the head of a LEX-arc emanating from j .)

(Syn.5) If i and j are individual nodes and m is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.5) m is the object of thought corresponding to the proposition that i is a (member of class) j .

(Syn.6) If i and j are individual nodes and m is an identifier not previously used, then



is a network and m is a structured proposition node.

(Sem.6) m is the object of thought corresponding to the proposition that (the class of) i s are (a subclass of the class of) j s.

(Syn.7) If i_1, i_2, i_3 are individual nodes, t_1, t_2 are temporal nodes, and m is an identifier not previously used, then

is a network and m is a structured proposition node.

(Sem.7) m is the object of thought corresponding to the proposition that agent i_1 performs act i_2 with respect to i_3 starting at time t_1 and ending at time t_2 , where t_1 is before t_2 .

(Sem.7) m is the object of thought corresponding to the proposition that agent i_1 performs act i_2 with respect to i_3 , starting at time t_1 and ending at time t_2 , where t_1 is before t_2 .

3. NATURAL-LANGUAGE UNDERSTANDING USING SEMANTIC NETWORKS

Semantic networks can be used for natural-language understanding as follows. The user inputs an English sentence to an augmented-transition-network (ATN) grammar (Woods, 1970, Shapiro 1982). The parsing component of the grammar updates a previously existing knowledge base containing semantic networks (or builds a new knowledge base, if there was none before) to represent the system's understanding of the input sentence. Note that this is *semantic* analysis, not *syntactic* parsing. The newly built node representing the proposition (or a previously existing node, if the input sentence repeated information already stored in the knowledge base) is then passed to the generation component of the ATN grammar, which generates an English sentence expressing the proposition in the context of the knowledge base. It should be noted that there is a *single* ATN parsing-generating grammar; the generation of an English output sentence from a node is actually a process of "parsing" the node into English. If the input sentence expresses a question, information-retrieval and inferencing packages are used to find or deduce an answer to the question. The node representing the answer is then passed to the generation grammar and expressed in English.

Here is a sample conversation with the SNePS system, together with the networks that are built as a result. User input is on lines with the :-prompt; the system's output is on the lines that follow. Comments are enclosed in brackets.

: Young Lucy petted a yellow dog
I understand that young Lucy petted a yellow dog

[The system is told something, which it now "believes." Its entire belief structure consists of nodes b1, m1-m13, and the corresponding sensory nodes (Figure 3). The node labeled "now" represents the current time, so the petting is clearly represented as being in the past. The system's response is "I understand that" concatenated with its English description of the proposition just entered.]

: What is yellow
a dog is yellow

[This response shows that the system actually has some beliefs; it did not just parrot back the above sentence. The knowledge base is not updated, however.]

: Dogs are animals
I understand that dogs are animals

[The system is told a small section of a class hierarchy; this information does update the knowledge base.]

[Figure 3 here]

There are three points to note about the use of SNePS for natural-language understanding. First, the system can "understand" an English sentence and express its understanding; this is illustrated by the first part of the conversation above. Second, the system can answer questions about what it understands; this is illustrated by the second part. Third, the system can incorporate new information into its knowledge base; this is illustrated by the third part.

Case Frames

Implicit in such a language understanding system are so-called *case frames*. We give a brief summary here; for a more thorough treatment see Winograd (1983).

Case-based deep structure analysis of English was suggested by Fillmore (1968). The surface structure of English relies only on the *order* of constituents and propositions in a clause to indicate role. Examples are:

Your dog just bit my mother.
My mother just bit your dog.

In Russian, Sanskrit, etc., explicit markings are used to represent relationships between participants. Examples in Russian, which uses six cases (nominative, genitive, dative, accusative, instrumental, and prepositional), are:

Professor uchenika tseloval (the professor kissed the student).
Professora uchenik tseloval (the student kissed the professor).

The extremely limited surface case system of English led Fillmore to suggest cases for English deep structure as follows: Agentive (animate instigator of action), Instrumental (inanimate force or object involved), Dative (animate being affected by action), Factive (object resulting from action), Locative (location or orientation), and Objective (everything else). For example, consider the sentence:

John opened the door with the key.

Its case analysis yields: Agentive = John, Objective = the door, Instrumental = the key.

Schank (1975) developed a representation for meaning (conceptual dependency) based on language independent conceptual relationships between objects and actions: case roles filled by objects (actor, object, attribuant, recipient), case roles filled by conceptualizations (instrument, attribute, ...), and case roles filled by other conceptual categories (time, location, state). For example:

John handed Mary a book.

has the analysis: Actor = John, Donor = John, Recipient = Mary, Object = book, Instrument = an action of physical motion with actor = John and object = hand.

4. SANSKRIT CASE FRAMES AND SEMANTIC NETWORKS

In the previous section we noted that natural-language understanding based on semantic networks involves determining what case frames will be used. The current set of case frames used in SNePS is not intended to be a complete set. Thus, we propose here that shastric Sanskrit case frames, implemented as SNePS networks, make an ideal knowledge-representation "language."

There are two distinct advantages to the use of classical Sanskrit analysis techniques. First, and of greatest importance, it is not an *ad hoc* method. As Briggs (1985) has observed, Sanskrit grammarians have developed a thorough system of semantic analysis. Why should researchers in knowledge representation and natural-language understanding reinvent the wheel? (cf. Rapaport 1986). Thus, we propose the use of case frames based on Sanskrit grammatical analysis in place of (or, in some cases, in addition to) the case frames used in current SNePS natural-language research.

Second, and implicit in the first advantage, Sanskrit grammatical analyses are easily implementable in SNePS. This should not be surprising. The Sanskrit analyses are case-based analyses, similar, for example, to those of Fillmore (1968). Propositional semantic networks such as SNePS are based on such analyses and, thus, are highly suitable symbolisms for implementing them.

As an example, consider the analysis of the following English translation of a Sanskrit sentence (from Briggs 1985):

Out of friendship, Maitra cooks rice for Devadatta in a pot over a fire.

Briggs offers the following set of "triples," that is, a linear representation of a semantic network for this sentence (Briggs 1985: 37, 38):

cause, event, friendship
friendship, object1, Devadatta

friendship, object2, Maitra
 cause, result, cook
 cook, agent, Maitra
 cook, recipient, Maitra
 cook, instrument, fire
 cook, object, rice
 cook, on-loc, pot

But what is the syntax and semantics of this knowledge-representation scheme? It appears to be rather *ad hoc*. Of course, Briggs only introduces it in order to compare it with the Sanskrit grammatical analysis, so let us concentrate on that, instead. Again using triples, this is:

cook, agent, Maitra
 cook, object, rice
 cook, instrument, fire
 cook, recipient, Devadatta
 cook, because-of, friendship
 friendship, Maitra, Devadatta
 cook, locality, pot

Notice that all but the penultimate triple begins with *cook*. The triple beginning with *friendship* can be thought of as a structured individual: the friendship between Maitra and Devadatta. Implemented in SNePS, this becomes the network shown in Figure 4. Node m11 represents the structured individual consisting of the relation of friendship holding between Maitra and Devadatta. Node m13 represents the proposition that an agent (named Maitra) performs an act (cooking) directed to an object (rice), using an instrument (fire), for a recipient (named Devadatta), at a locality (a pot), out of a cause (the friendship between the agent and the recipient).

Such an analysis can, presumably, be algorithmically derived from a Sanskrit sentence and can be algorithmically transformed back into a Sanskrit sentence. Since an English sentence, for instance, can also presumably be analyzed in this way (at the very least, sentences of Indo-European languages should be easily analyzable in this fashion), we have the basis for an interlingual machine-translation system grounded in a well-established semantic theory.

[Figure 4 here]

5. INTERLINGUAL MACHINE TRANSLATION

The possibility of translating natural-language texts using an intermediate common language was suggested by Warren Weaver (1949). Translation using a common language (an "interlingua") is a two-stage process: from source language to an interlingua, and from the interlingua to the target language (Figure 5). This approach is characteristic of a system in which representation of the "meaning" of the source-language input is intended to be independent of any language, and in which this same representation is used to synthesize the target-language output. In an alternative approach (the "transfer" approach), the results of source text analysis are converted into a corresponding representation for target text, which is then used for output. Figure 6 shows how the interlingua (indirect) approach compares to other (direct and transfer) approaches to machine translation (MT). The interlingua approach to translation was heavily influenced by formal linguistic theories (Hutchins 1982). This calls for an interlingua to be formal, language-independent, and "adequate" for knowledge representation.

[Figures 5 and 6 here]

Various proposals for interlinguas have included the use of formalized natural-language, artificial "international" languages like Esperanto, and various symbolic representations. Most prior work on interlinguas has centered on the representation of the lexical content of text. Bennet et al. (1986) point out that a large portion of syntactic structures, even when reduced to "canonical form," remain too language-specific to act as an interlingua representation. Thus, major disadvantages of an interlingua-based system result from the practical difficulty of actually defining a language-free interlingua representation.

Besides, none of the existing MT systems use a significant amount of semantic information (Slocum 1985). Thus, the success of an interlingua depends on the nature of the interlingua as well as the analysis rendered on the source text to obtain its interlingual representation. This made the interlingua approach too ambitious, and researchers have inclined more towards a transfer approach.

It has been argued that analyses of natural-language sentences in semantic networks and in Sanskrit grammar is remarkably similar (Briggs 1985). Thus we propose an implementation of Sanskrit in a semantic network to be used as an interlingua for MT. As an interlingua, Sanskrit fulfills the basic requirements of being formal, language-independent, and a powerful medium for representing meaning.

REFERENCES

- [1] Bennet, P. A.; Johnson, R. L.; McNaught, John; Pugh, Jeanette; Sager, J. C.; and Somers, Harold L. (1986), *Multilingual Aspects of Information Technology*, (Hampshire, Eng: Gower).
- [2] Brachman, Ronald J., & Levesque, Hector J. (1985), *Readings in Knowledge Representation*, (Los Altos, CA: Morgan Kaufmann).
- [3] Brachman, Ronald J., & Schmolze, James G. (1985), "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, 9: 171-216.
- [4] Briggs, Rick (1985), "Knowledge Representation in Sanskrit and Artificial Intelligence," *AI Magazine*, 6.1 (Spring 1985) 32-39.
- [5] Fillmore, Charles (1968), "The Case for Case," in E. Bach & R. T. Harms (eds.), *Universals in Linguistic Theory*, (Chicago: Holt, Rinehart and Winston): 1-90.
- [6] Fine, K. (1983), "A Defence of Arbitrary Objects," *Proc. Aristotelian Soc.*, Supp. Vol. 58: 55-77.
- [7] Findler, Nicholas V. (1979), *Associative Networks: The Representation and Use of Knowledge by Computers*, (New York: Academic Press).
- [8] Gelb, Ignace J. (1985), Linguistics, *Encyclopedia Britannica Macropedia*, Fifteenth Edition.
- [9] Hutchins, John W. (1982), "The Evolution of Machine Translation Systems," in V. Lawson (ed.), *Practical Experiences of Machine Translation*, (Amsterdam: North-Holland): 21-38.
- [10] Maida, Anthony S., & Shapiro, Stuart C. (1982), "Intensional Concepts in Propositional Semantic Networks," *Cognitive Science*, 6: 291-330; reprinted in Brachman & Levesque 1985: 169-89.
- [11] McKay, D. P., & Shapiro, Stuart C. (1981), "Using Active Connection Graphs for Reasoning with Recursive Rules," *Proc. IJCAI-81*, 368-74.
- [12] Rapaport, William J. (1986), "Logical Foundations for Belief Representation," *Cognitive Science*, 10, 371-422.
- [13] Schank, Roger C. (1975), *Conceptual Information Processing*, (Amsterdam: North Holland), 1975.
- [14] Shapiro, Stuart C. (1978), "Path-Based and Node-Based Inference in Semantic Networks," in D. Waltz (ed.), *Tinlap-2: Theoretical Issues in Natural Language Processing*, (New York: ACM): 219-25.
- [15] Shapiro, Stuart C. (1979), "The SNePS Semantic Network Processing System," in Findler 1979: 179-203.

- [16] Shapiro, Stuart C. (1982), "Generalized Augmented Transition Network Grammars For Generation From Semantic Networks," *American Journal of Computational Linguistics*, 8: 12-25.
- [17] Shapiro, Stuart C.; Martins, J.; & McKay, D. P. (1982), "Bi-Directional Inference," *Proc. 4th Annual Conf. Cognitive Science Soc.*, (U. Michigan): 90-93.
- [18] Shapiro, Stuart C., & McKay, D. P. (1980), "Inference with Recursive Rules," *Proc. AAAI-80*, 151-53.
- [19] Shapiro, Stuart C., & Rapaport, William J. (1986), "SNePS Considered as a Fully Intensional Propositional Semantic Network," in G. McCalla & N. Cercone (eds.), *The Knowledge Frontier*, (New York: Springer-Verlag), 262-315.
- [20] Shapiro, Stuart C., & Rapaport, William J. (1986), "SNePS Considered as a Fully Intensional Propositional Semantic Network," *Proc. National Conference on Artificial Intelligence (AAAI-86; Philadelphia)*, Vol. 1 (Los Altos, CA: Morgan Kaufmann): 278-83.
- [21] Slocum, Jonathan (1985), "A Survey of Machine Translation: its History, Current Status, and Future Prospects," *Computational Linguistics*, 11: 1-17.
- [22] Srihari, Rohini K. (1981), "Combining Path-Based and Node-Based Inference in SNePS," Tech. Rep. 183, (SUNY Buffalo Dept. of Computer Science).
- [23] Weaver, Warren (1949), "Translation," in W. N. Locke & A. D. Booth (eds.), *Machine Translation of Languages*, (Cambridge, MA: MIT Press, 1955): 15-23.
- [24] Winograd, Terry (1983), *Language as a Cognitive Process*, Reading, MA: Addison-Wesley.
- [25] Woods, William (1970), Transition Network Grammars for Natural Language Analysis, *Communications of the ACM*, 13: 591-606.

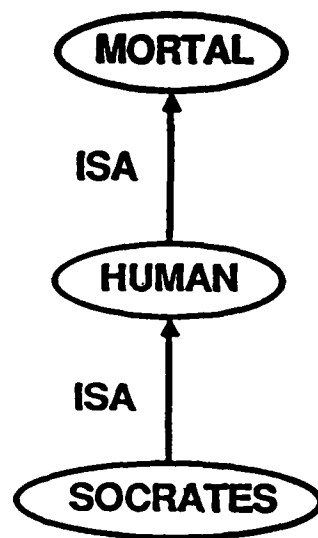


Figure 1(a). An "ISA" inheritance-hierarchy semantic network

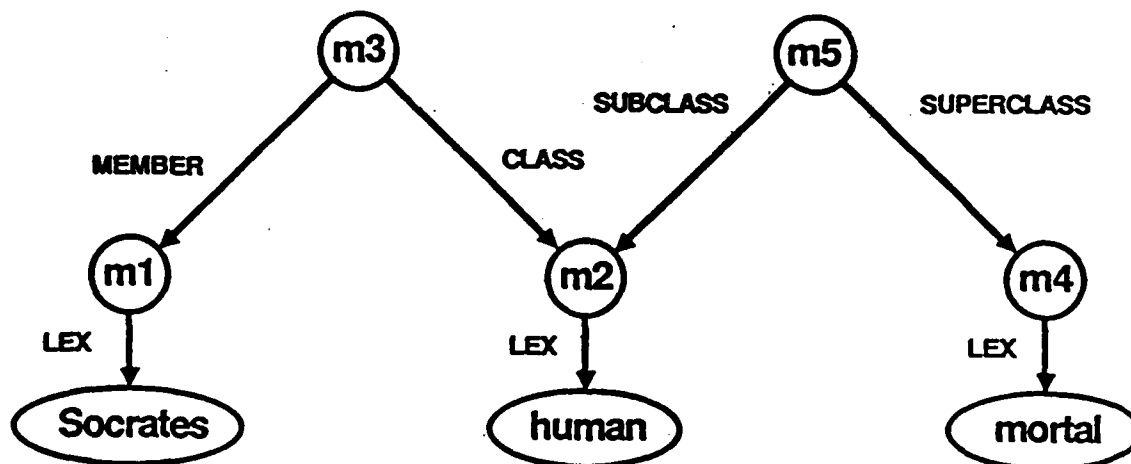


Figure 1(b). A SNePS propositional semantic network (m3 and m5 represent the propositions that Socrates is human and that humans are mortal, respectively)

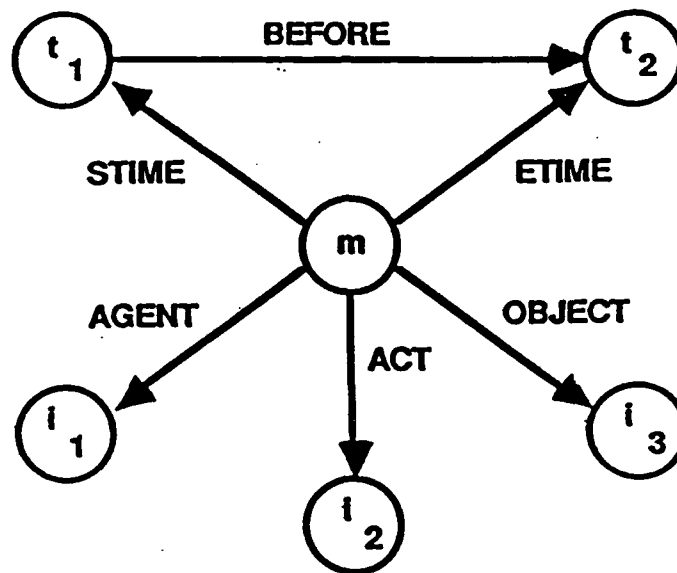


Figure 2.

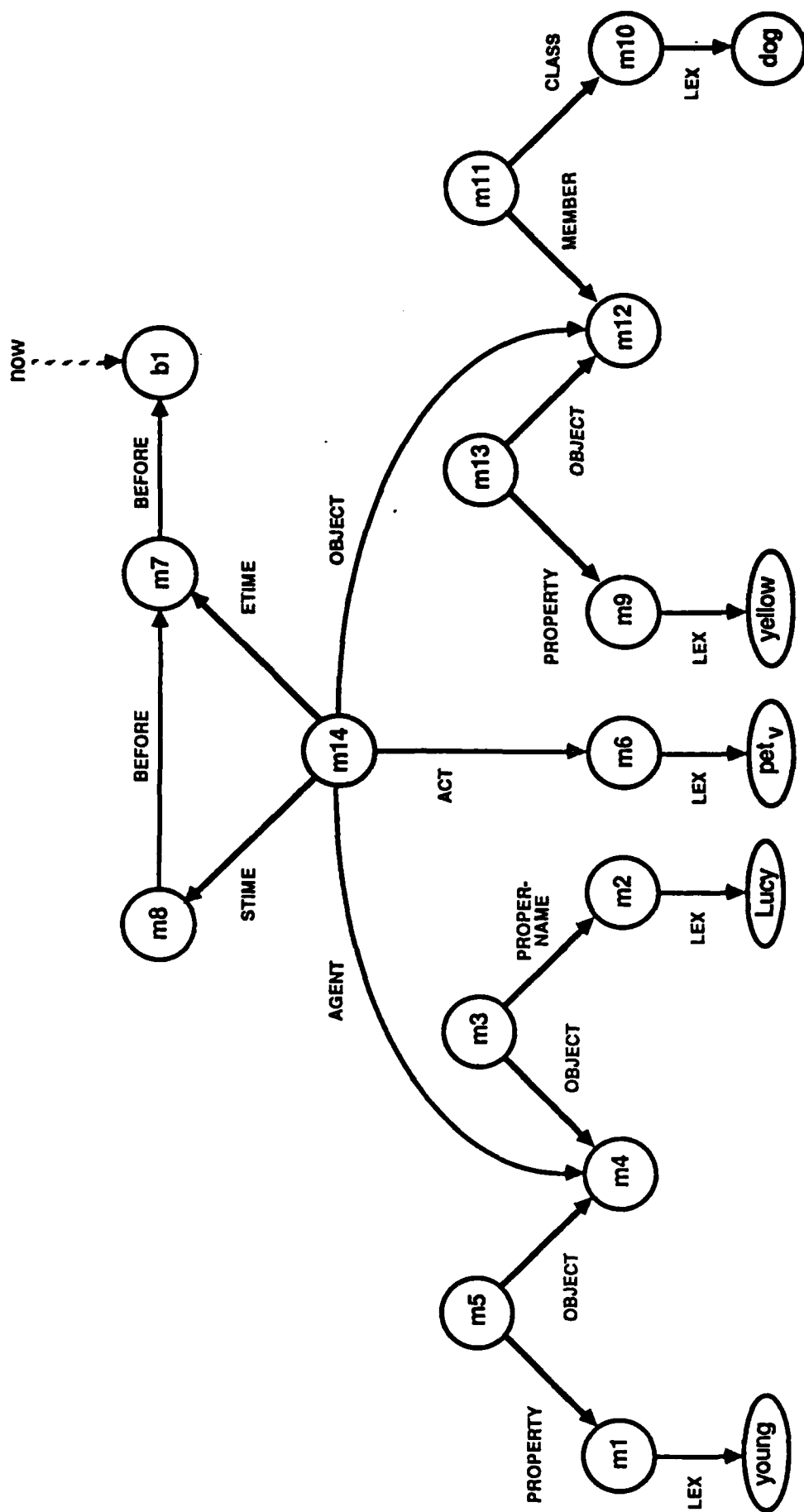


Figure 3. SNePS network for "Young Lucy petted a yellow dog."

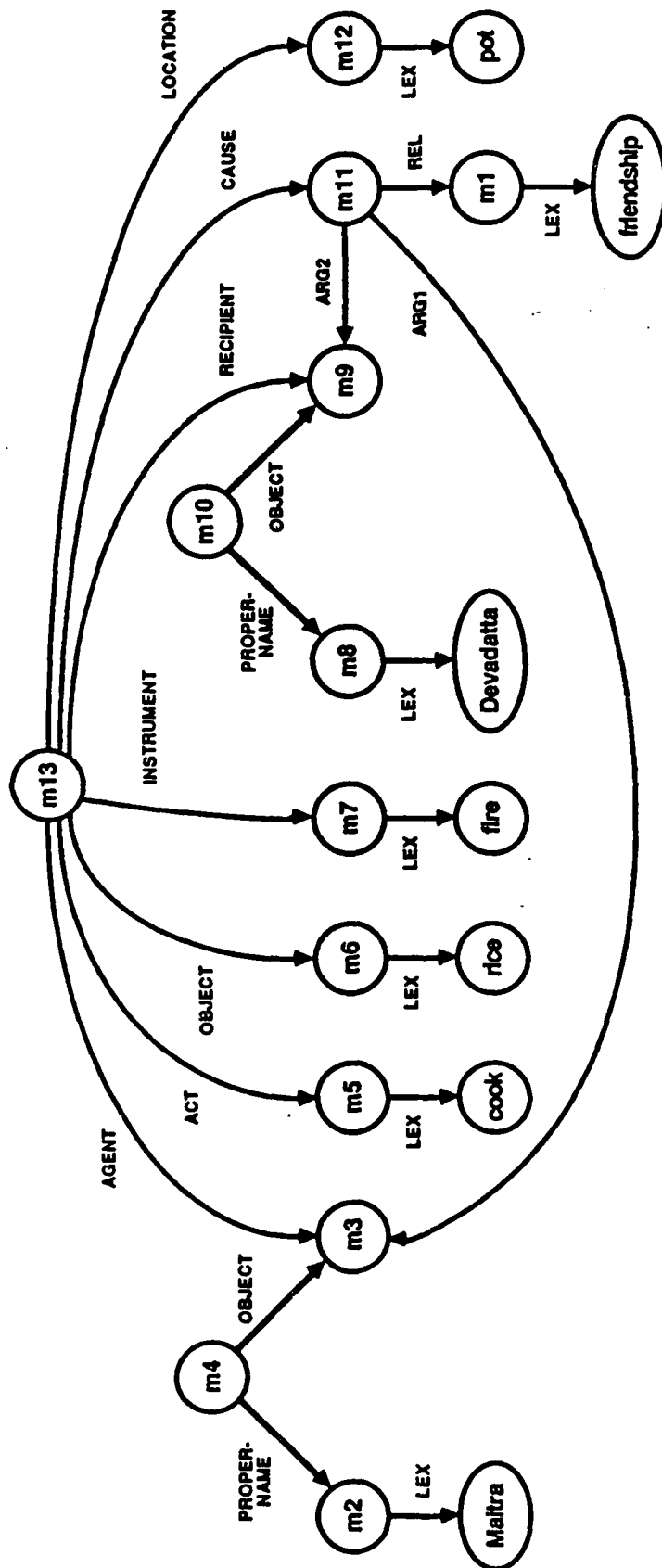


Figure 4. SNePS network for "Out of friendship, Maitra cooks rice for Devadatta in a pot over a fire."

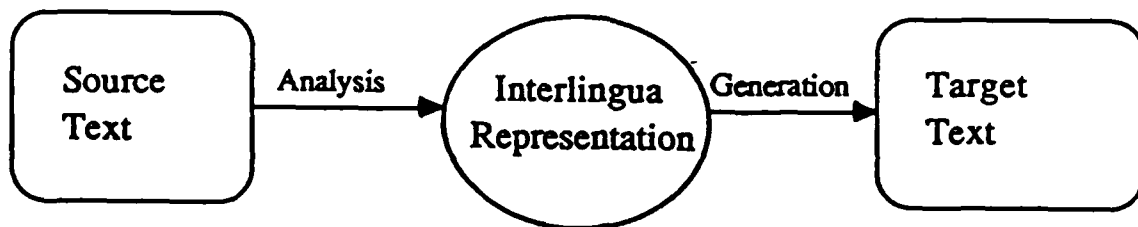


Figure 5. The Interlingua Approaches

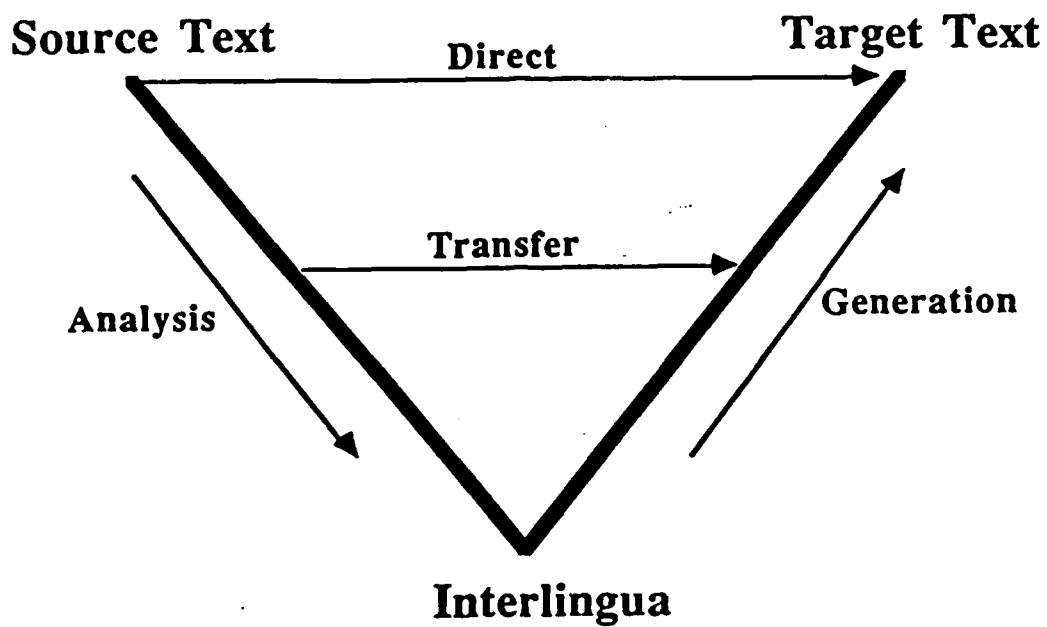


Figure 6. Various Approaches to MT

Appendix B2:

"Knowledge-Based Parsing"

Neal, J.G.
Shapiro, S.C.

Knowledge-Based Parsing

J.G. Neal and S.C. Shapiro

Abstract. An extremely significant feature of any natural language (NL) is that it is its own metalanguage. One can use an NL to talk about the NL itself. One can use an NL to tutor a non-native speaker, or other poor language user, in the use of the same NL. We have been exploring methods of knowledge representation and NL understanding (NLU) which would allow an artificial intelligence (AI) system to play the role of poor language user in this setting. The AI system would have to understand NL utterances about how the NL is used, and improve its NLU abilities according to this instruction. It would be an NLU system for which the domain being-discussed in NL is the NL itself.

Our NLU system is implemented in the form of a general rule-based inference system which reasons according to the rules of its knowledge base. These rules comprise the system's knowledge of language understanding in the same way that the rules of any rule-based system comprise that system's knowledge of its domain of application. Our system uses the same knowledge base for both linguistic and other knowledge since we feel that there is no clear boundary line separating syntactic, semantic, and world knowledge.

We are exploring the possibility of an NLU system's becoming more facile in its use of some language by being told how that language is used. We wish this explanation to be given in an increasingly sophisticated subset of the language being taught. Clearly, the system must start with some language facility, but we are interested in seeing how small and theory-independent we can make the initial, "kernel" language. This article reports the current state of our work.

1 Introduction

1.1 Overview

An extremely significant feature of any natural language (NL) is that it is its own metalanguage. One can use an NL to talk about the NL itself. One can use an NL to tutor a non-native speaker, or other poor language user, in the use of the same NL. We have been exploring methods of knowledge representation (KR) and NL understanding (NLU) which would allow an artificial intelligence (AI) system to play the role of poor language user in this setting. The AI system would have to understand NL utterances about how the NL is used, and improve its NLU abili-

ties according to this instruction. It would be an NLU system for which the domain being discussed in NL is the NL itself.

It is essential to our approach to have the system's parsing and linguistic knowledge be an integral part of its domain knowledge. Acknowledging that what is meant by "meaning" is controversial (Quine, 1948), we take the meaning or significance of a word or phrase to include linguistic knowledge about the word or phrase. For example, we feel that how a word like "dog" is used in language is a part of its "meaning", along with other properties such as the fact that "dog" denotes a special kind of animal with typical characteristics. The implementation of our system is based upon the above stated view and therefore the rules and assertions comprising the system's knowledge of language understanding, including syntax, is integrated into the system's knowledge base along with its other task domain knowledge.

We are exploring the possibility of an NLU system's becoming more facile in its use of some language by being taught how that language is used. The teacher might be a conversation partner who happens to use some phrase the system is not yet familiar with, or a language theorist who wants to find out if she can explain her theory completely and clearly enough for the system to use language according to it. We wish this explanation to be given in an increasingly sophisticated subset of the language being taught. That is, why not test and make use of the system's language capability by using it to continue the system's "education"? Clearly, the system must start with some language facility, but we are interested in seeing how small and theory-independent we can make the initial, "kernel" language.

In this chapter, we will discuss our knowledge representation techniques, the system's kernel language (KL), and parsing strategy. We will demonstrate how our system can be instructed in the use of some language defined by the teacher and how the system's acquired language can itself be used as its own metalanguage. The kernel language only incorporates primitive relations such as one token being a predecessor of another in a string, membership in a lexical or string category, and constituency. As an example of using the system's language as its own metalanguage to enhance its language capability, we will demonstrate, starting with only the KL, how the system can be instructed with regard to the number (i.e., singular or plural) of some words and then be informed that "If the head-noun of a noun-phrase X has number Y, then X has number Y". This newly acquired knowledge can then be applied by the system to infer that since "glasses" is plural, so is "the old man's glasses" when it reads this phrase in a sentence such as "The old man's glasses were filled with water".

Our system is able to understand when strings are *mentioned* in input utterances as well as when they are *used* to communicate with the system. This capability is demonstrated frequently in this chapter, but particularly in Sect. 4 with the classic sentence from Tarski (1944): "'Snow is white' is true if and only if snow is white".

The use of inference and world knowledge is essential for a system to parse sentences such as "John saw the bird without binoculars" and "John saw the bird without tailfeathers" from Schubert and Pelletier (1982) or "John saw the man on the hill with a telescope". Our research is based upon the concept of having parsing performed by a general reasoning system which has the capability of applying

world knowledge inferences during parsing, since the "parser" is not a separate isolated component with special sublanguage, representations, or knowledge base.

1.2 Fundamental Assumptions

Our system incorporates the *use-mention distinction* (Quine, 1951) for language. Our representations reflect the fact that the meaning of a token or surface string is distinct from the token or string itself. Our system's knowledge base maintains a representation for a token or surface string that is distinct from the representation of the interpretation of the input token or string. This distinction is the same as between a numeral and a number in mathematics. To refer to a word or string rather than its meaning, the user must use the usual English convention of prefacing the word by a single-quote mark or enclosing the string in quotation marks. (See Sections 2.2.1 and 2.4.2 for more information.)

A second principle upon which our work is based is that each occurrence of a given surface string in the input stream is assumed to have a different interpretation, unless the teacher has entered rules into the system to dictate otherwise. For example, if a name such as "John" has been entered into the lexicon and is used twice, either in successive utterances or within the same utterance, then the system interprets each occurrence of the name as referring to a different entity unless the teacher has instructed the system otherwise. Since an NLU system must be capable of handling ambiguities, and, in a situation in which no explicit rules are known to the system to guide it in determining whether a word or phrase is ambiguous, it must have a default procedure to follow, we have chosen to implement the above principle. Although our approach would seem to overly complicate the network, it is a reasonable default principle since there is some evidence that merging of nodes is easier than splitting nodes (Maida and Shapiro, 1982).

A third principle which is fundamental to our theory is that all possible parses and interpretations of a surface string are to be determined according to the language definition used by the system. We feel that multiple interpretations, when justified by the language definition, are warranted since agile human minds frequently perceive alternative interpretations and, in fact, a great deal of humor is dependent upon this.

Our system does not currently do morphological analysis. One of the areas in which we plan to do future research is knowledge-based morphological analysis. We plan to develop a system component that would perform morphological analysis and function as a preprocessor or coprocessor with the system discussed in this article.

1.3 Declarative Knowledge Representation in an Integrated Knowledge Base

Our approach is to represent knowledge in declarative form, to the greatest extent possible, in the semantic network formalism. This applies to all knowledge including linguistic knowledge and the rules which are applied by the inference machine

to guide the system's reasoning, the parsing process being one manifestation of the system's reasoning according to the rules of its network knowledge base. It is our intent that the system's knowledge, including its linguistic knowledge, be available to the teacher in the same way that domain knowledge is in other AI systems.

Furthermore, the declarative form is a more suitable form for linguistic knowledge in theoretical studies of language. A language definition or description is inherently declarative, and as Pereira and Warren have pointed out: "The theorists have concentrated on describing *what* natural language is, in a clear and elegant way. In this context, details of *how* natural language is actually recognized or generated need not be relevant, and indeed should probably not be allowed to obscure the language definition" (1980, p.269, italics in the original). In this regard, a declarative representation is preferable to a formalism such as an ATN, in that the ATN is a description of a *process* for recognizing a language.

Our system uses an integrated knowledge base for both linguistic and other knowledge as advocated by Pollack and Waltz (1982) and by Dahl (1981). As indicated in Section 1.1, we take the meaning of a word or phrase to include linguistic knowledge about the word or phrase and its use. Furthermore, we feel that there is no clear boundary line separating syntactic, semantic, and world knowledge. For example, it is not clear to what extent the classification of words into lexical categories depends on meaning, function, or form. Should certain words be classified as mass nouns because they fit certain distributional frames or have a certain form (e.g., I used {sand, the sand, a bag of sand, *a sand, *two sands.}) or are the frames and forms simply a reflection of the property we think of as characterizing the substances named by mass nouns, namely that the substance is not naturally physically bounded and that when two amounts of the substance are "put together" they become one amount? Perhaps certain aspects of syntax cannot or should not be separated from semantics. Furthermore, the terms "semantic knowledge" and "world knowledge" seem only to be used to informally express a measure of the sophistication or complexity of knowledge.

1.4 System Overview

Our Natural Language System is being developed and implemented using the SNePS semantic network processing system (Shapiro, 1979a; Shapiro and the SNePS Group, 1981). The terminology and representations for some of the basic categories, objects, and relations of this work evolved from a preliminary study reported in Shapiro and Neal (1982). Figure 1 illustrates an overview of the system.

The semantic network formalism has been used by many researchers for knowledge representation (Quillian, 1968, 1969; Rumelhart and Norman, 1973; Simmons, 1973; Woods, 1975; P. Hayes 1977; Schubert, 1976; Hendrix, 1978, 1979; Schubert et al., 1979; Brachman, 1979). In contrast to other semantic network implementations, the SNePS system provides a uniform declarative representation for both rules and assertions in the network (Shapiro, 1971, 1979b). Furthermore, our system comprises an effort to utilize a common representation for problem-solving and language-comprehension information as advocated by Charniak

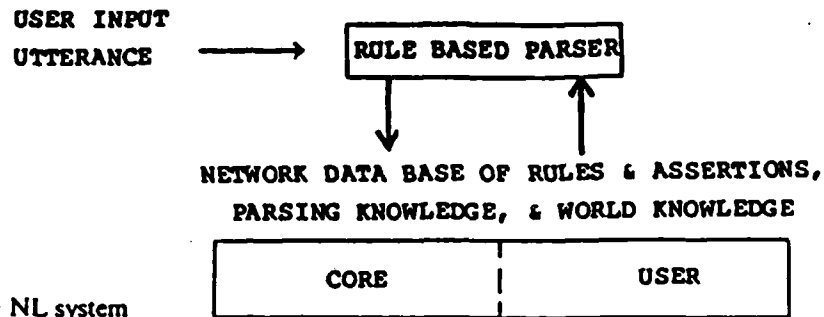


Fig. 1. Overview of the NL system

(1981). Our system is similar to the Prolog-based (Roussel, 1975) systems of Warren and Pereira (1982), Dahl (1979, 1981), and McCord (1982) in that it is implemented in a logic-based system in which processing is a form of inference. The SNePS inference package (Shapiro et al., 1982), however, is not based on the resolution principle (Robinson, 1965) as is Prolog, but on a multi-processing approach (Kaplan, 1973; McKay and Shapiro, 1980) incorporating a producer-consumer model. SNePS also provides a facility for "procedural attachment" in rules to handle processing knowledge for which the declarative network representation is unnatural.

The PSI-KLONE system (Bobrow and Webber, 1980) uses linguistic knowledge represented in a KL-ONE network (Brachman, 1978a, 1978b, 1979) to function as semantic interpreter for parsed surface strings. The PSI-KLONE interpreter, however, functions in cooperation with an ATN parser in the RUS framework (Bobrow, 1978). In contrast, we are implementing an integrated system for syntactic and semantic processing which uses a uniform representation for syntactic and semantic knowledge.

The rule-based parser of Figure 1 is essentially the SNePS inference package which reasons according to the rules of the knowledge base.

The knowledge base consists of CORE knowledge and USER knowledge. The CORE knowledge is provided by the designers of the system and defines a kernel language initially acceptable to the system. USER knowledge results from the processing of user input utterances.

The function of our NL parser is twofold:

1. derivation of zero or more annotated parse trees for the input surface string;
2. construction of a network representation for the interpretation of the input utterance from the annotated parse tree and from other relevant knowledge from the network data base.

The above two functions are not handled by separate processors, but, instead, are both accomplished by the SNePS inference package as a result of the application of CORE and USER rules. The processes of accomplishing the two functions are interrelated and can cooperate. The interpretation of a surface string will depend on how it is syntactically parsed and, conversely, the syntactic parse of a surface string can depend on the meanings of related, constituent, or neighboring strings.

The two processes are not carried out in a purely sequential fashion for a given input utterance, since interpretations can be constructed for parsed constituent strings before the parsing of the entire utterance is complete.

1.5 Knowledge Representation Techniques

A SNePS semantic network is a directed graph with labeled arcs in which nodes represent concepts and the arcs represent nonconceptual binary relations between concepts. It is generally agreed that the nodes of a semantic network represent intensional concepts (Woods, 1975; Brachman, 1977; Maida and Shapiro, 1982). A "concept" is something in our domain of interest about which we may want to store information and which may be the subject of "thought" and inference. Since each concept is represented by a node, the relations represented by the arcs of our system are not conceptual, but structural (Shapiro, 1979a).

The primary type of arc in a SNePS network is the *descending* arc and if there is a path of descending arcs from node N to node M, N is said to *dominate* M. Two important types of nodes are *molecular* and *atomic* nodes. Molecular nodes are nodes that dominate other nodes. Atomic nodes are simply not molecular. Atomic nodes can be *constant* (representing a unique semantic concept) or *variable*. Variable nodes are used in SNePS as variables are used in normal predicate logic notations. Network nodes can also be categorized as in the table in Figure 2.

A propositional molecular node N together with the arcs incident from the node and the nodes M_1, \dots, M_k immediately dominated by N correspond to a case frame (Fillmore, 1968; Bruce, 1975) where the arc names correspond to the slot names, and the nodes M_1, \dots, M_k represent the slot fillers. Undominated molecular nodes in a SNePS network represent propositions believed by the system. Concepts such as the following are propositional and are represented by molecular nodes: Lexeme L is a member of category C; S1 is a constituent string of S2; lexeme L has number N (i.e. singular or plural). Simple examples of propositional nodes are M1 and M2 of Figure 3.

Node M1 represents the proposition that B1 represents the concept expressed by the word "NOUN" and M2 represents the proposition that the lexeme "SNOW" is in the category called "NOUN".

The syntactic objects represented in our network knowledge base include morphemes, surface strings, and nodes of annotated parse trees. Individual morphemes are represented as nodes whose identifiers or print names are the morphemes themselves. The representation of a surface string utilized in this study consists of a network version of the list structure used by Pereira and Warren (1980). This representation is also similar to Kay's charts (1973) in that whenever alternative analyses are made for a given substring of a sentence, the sentence structure is enhanced by multiple structures representing these alternative analyses. Retention of the alternatives avoids the reanalyses of previously processed substrings which occurs in a backtracking system. Our basic representation of a surface string is illustrated in Figure 4.

Nodes identified by the atoms B0, SNOW, IS, and WHITE are atomic nodes and represent objects: the empty string, and tokens "SNOW", "IS", and

<u>Node Category</u>	<u>Type of Concept</u>
Non-dominated (asserted) molecular node	Asserted proposition which is "believed" by the System
Dominated molecular node	Proposition or structured object which is a participant in a proposition
Atomic node	Object

Fig.2. Table of node categories

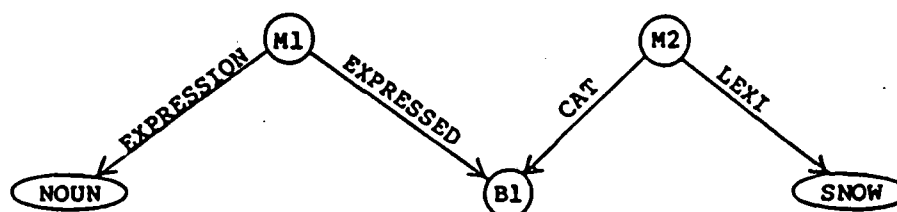


Fig.3. M1 and M2 are simple examples of propositional nodes

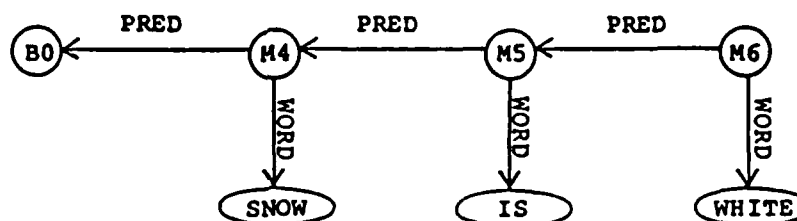


Fig.4. Basic network representation of a surface string

"WHITE", respectively. Node M4 is molecular and represents the initial string "SNOW". M5 is also molecular and represents the initial string "SNOW IS", and similarly for node M6. A node such as M6 that represents an object would typically be dominated in our system by some node representing a proposition about it.

As each word of an input string is read by the system, the network representation of the string is extended and relevant rules stored in the SNePS network are triggered.

Interpretations of surface strings are also represented as nodes of our network knowledge base. The kernel language of the system enables the user to define case frame structures and to define rules to guide the system in interpreting input utterances.

1.6 Core Knowledge and the Kernel Language

Our approach is to provide the teacher (user) with a kernel language in which she can begin to "explain" the syntax and semantics of some natural or invented language to the system. The present version of our kernel language includes:

- a) predefined terms such as L-CAT, the set of the names of lexical classes, and S-CAT, the set of the names of string classes; S-CAT contains the important category names ANT-CLAUSE, CQ-CLAUSE, and RULE-STMT, which are used to bootstrap into a more sophisticated rule input capability;
- b) predefined objects such as (i) initial strings and (ii) bounded strings with beginning and ending token;
- c) predefined relations such as (i) lexeme L is a member of category C; (ii) bounded string B is a member of category C and this structure is represented by S; (iii) structure S expresses concept C; (iv) structure S1 is a constituent of structure S2;
- d) predefined functions such as (i) a test to determine whether two network nodes are identical, (ii) a test to determine whether two bounded surface strings match, and (iii) a test to determine whether one bounded string precedes another bounded string.

The KL provides the teacher with a basic language of rewrite rules for the purpose of defining *syntactic* lexical insertions, context free phrase structure rules, and context sensitive rules as well as *semantic* mappings from string categories to case frame structures and mappings from string categories to case frame participant or component slots.

1.7 Metalanguage Conventions and Symbols

In this chapter, we use the notational convention that words written in upper case letters denote words of KL and we use the metasymbols:

- () denote a non-terminal; if the angle brackets enclose the name of a category of the core or a user-defined category, then such usage denotes a variable whose domain is the category named within the enclosing brackets.
- () for grouping.
- * Kleene star: when used as a superscript on an item, denotes zero or more of the items in a finite sequence.
- + when used as a superscript on an item, denotes one or more of the items in a finite sequence.
- ... ellipsis.

2 Core Knowledge and Representations

2.1 Uniform Representation and Intensional Constructs

We use the semantic network formalism to represent both syntactic and semantic knowledge in the form of assertions and rules to be applied in inference formation. We include linguistic knowledge in the network knowledge base and use the network formalism as a uniform "language" with which to represent all types of knowledge. Thus we model surface strings and syntactic properties and categories as intensions (Woods, 1975; Brachman, 1977; Maida and Shapiro, 1982), concepts, or objects of thought.

2.2 Predefined Categories, Objects, Relations, Functions

2.2.1 Predefined Categories

We are investigating the capability of an NLU system becoming more adept in the use of some language by being instructed in the use of the language. The system must start with some language facility, but we are striving to make the core knowledge base as small and theory-independent as possible.

Included among the core primitives are certain predefined categories. Since we are designing a language *capability* that is as theory-independent as possible and not a robust parser for a predetermined language such as English, some of these categories are initially empty, while others have very few members. All the categories are to be utilized by the teacher, either directly or indirectly, and the membership of the categories expanded by the teacher as the definition of her target language takes shape.

The most basic of these categories are L-CAT, S-CAT, and VARIABLE. L-CAT consists of the names of lexical classes or classes of terms. L-CAT initially contains the predefined terms L-CAT, S-CAT, VARIABLE, PUNCTUATION, and FUNCTION-NAMES. Names that the teacher would add to L-CAT might include, for example, NOUN, VERB, and PREPOSITION.

The purpose of VARIABLE is to contain all the identifiers that the teacher will use as variables in her processing rules when stated as input to the system. The VARIABLE category is initially empty.

The category PUNCTUATION initially contains the punctuation marks period, single-quote, and double-quote.

The class FUNCTION-NAMES contains the names of the functions that the teacher has available to be used in a form of procedural attachment to the declarative rules of the network knowledge base. FUNCTION-NAMES initially contains the names of the tests discussed in Section 2.2.4: IDENTITY-TEST, STRING-MATCH-TEST, and PRECEDES-TEST.

S-CAT is defined to be the set of all the names of string categories. S-CAT initially contains the names of the predefined string categories UTTERANCE, P-RULE, CASE-FRAME-DEFINITION, CASE-SLOT-DEFINITION, LITERAL, LITERAL-STRING, UNIQUE-MEANING-CAT, VAR-APPOSITION-PHR,

MAIN-APPOS-PHR, VAR-NAME, ANT-CLAUSE, CQ-CLAUSE, and RULE-STMT. The string categories P-RULE, CASE-FRAME-DEFINITION, CASE-SLOT-DEFINITION, LITERAL, LITERAL-STRING, and VAR-NAME each have predefined syntax. For the remaining string categories except UTTERANCE, the definition of the syntax is left to the teacher, VAR-APPOSITION-PHR and RULE-STMT having restrictions discussed later in this article. The class UTTERANCE contains all input surface strings.

The predefined string category P-RULE includes all strings that qualify as production or syntactic rewrite rules as discussed in Section 2.5.2. These rewrite rules are part of the kernel language understood by the system.

The kernel language includes semantic rewrite rules to enable the teacher to define case frames and associations between case frames and particular string categories for use in the interpretation of input utterances. CASE-FRAME-DEFINITION and CASE-SLOT-DEFINITION are string categories that contain the two types of semantic rewrite rules. The capability associated with the CASE-FRAME-DEFINITION and CASE-SLOT-DEFINITION classes is discussed in Section 2.5.3.

LITERAL is the category of strings consisting of a single-quote mark followed by a lexeme. LITERAL-STRING is the category of strings that consist of a pair of double-quote marks enclosing a surface string.

VAR-APPOSITION-PHR, MAIN-APPOS-PHR, and VAR-NAME are string categories that enable a variable to be used as an appositive so as to establish the variable as the identifier for the MAIN-APPOS-PHRase which it is adjacent to. For example, after input of appropriate user-defined rules to the system, the string "a noun-phrase X" (from the sentence "If the head-noun of a noun-phrase X has number Y then X has number Y") could be parsed as a VAR-APPOSITION-PHR with "a noun-phrase" as the MAIN-APPOS-PHR and "X" the VAR-NAME so that in parsing the stated rule, X is remembered by the system as an identifier for the unknown noun-phrase referred to in the phrase and is thus capable of being referred to again later as in the given rule example. Since no referencing mechanisms are built into our system to enable the teacher to refer to previously mentioned concepts, the above string categories assist the teacher in establishing rules to determine the referencing process according to her own theory. The use of this capability is illustrated by example in Section 3.

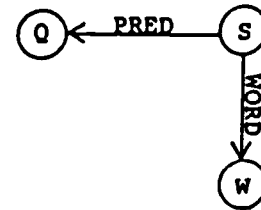
As the teacher proceeds to instruct the system in her language definition, she will need to enter rules that cannot be expressed in the language of rewrite rules. Such rules would include rules concerning the semantics of utterances. Therefore, the core primitives include three initially empty string categories, RULE-STMT, ANT-CLAUSE, and CQ-CLAUSE to enable the teacher to define the syntax of general conditional rules. These categories are discussed in subsequent sections.

UNIQUE-MEANING-CAT is defined to be the class of all the strings that have a unique meaning. That is, if a string is in UNIQUE-MEANING-CAT, it must express the same intension each time it is encountered in an input utterance. As stated in Section 1.2, a premise of our theory and NL system is that each time a given word or string is "read" by the system, it has a new or different meaning unless this meaning is determined by rules and/or assertions input by the teacher.

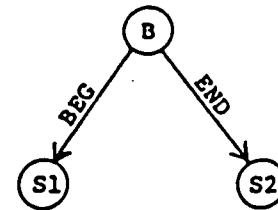
2.2.2 Predefined Objects

The predefined objects essential to our theory and implementation are the concepts of the Initial String and the Bounded String. These objects and their network representations are described below.

- a) Initial string S consists of the word or symbol W concatenated to the initial string Q. Q may be the null string represented by node B0.



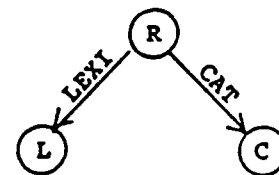
- b) Bounded string B represents the surface string beginning with the last word of initial string S1 and ending with the last word of initial string S2 where S1 precedes S2.



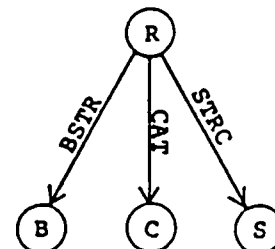
2.2.3 Predefined Relations

It is necessary for the NL system to have a set of predefined relations for knowledge representation. The current set of these relations and their corresponding semantic network structures are listed below.

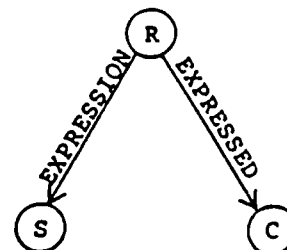
- a) Lexeme L is a member of category C; e.g., node M21 of Figure 5 represents the concept that 'STUDENT is a NOUN.



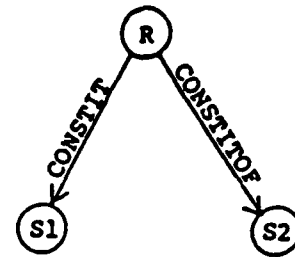
- b) The bounded string B is in category C and this structure or parse of the string B is represented by node S (analogous to a node of an annotated parse tree); e.g., node M43 of Figure 5 represents the concept that the structure represented by B21 represents a parsing of the bounded string represented by M42 as an INDEF-NOUN-PHRASE.



- c) Structure or parsed string S expresses concept C; e.g., node M20 of Figure 5 represents the concept that the string "NOUN" expresses the category of nouns represented by node B10.



- d) The structure S1 is a constituent of structure S2; e.g., node M44 of Figure 5 represents the concept that the literal 'STUDENT is a constituent of the structure represented by node B21.



- e) The rule structures of SNePS (Shapiro, 1979 a).

Figure 5 shows a surface string enhanced by additional structure that would result from the system's reading and parsing the input string "A STUDENT" after some syntactic rules had been input by the teacher (e.g., 'A is an INDEF-DET, 'STUDENT is a NOUN, a string consisting of an INDEF-DET followed by a NOUN is an NOUN-PHRASE).

2.2.4 Predefined Functions

The following functions are essential for the NL system and could not be efficiently implemented in the declarative SNePS language.

- Identity test* takes two network nodes as arguments, and returns true if the two nodes are identical and returns false otherwise.
- String-match test* takes two bounded strings in network representation as arguments, and returns true if the sequence of words or symbols in the two strings are identical, and returns false otherwise.
- Precedes test* takes two bounded strings in network representation as arguments, and returns true if the first string precedes the second string in the input stream, and returns false otherwise.

2.3 The Reading Function

The system's reading function "reads" one token (lexeme or punctuation mark) at a time from the input stream. For each input token, the structure of Figure 6 is added to the network, where node S represents the previously added initial string, C represents the lexical category of the token, I represents the newly established initial string, and B represents the newly added bounded string.

If the token belongs to any lexical categories, this membership would already be represented in the network in the form of relation (a) of Section 2.2.3 (how such relations are established is explained in Section 2.4). The lexical categories to which the input token belongs are found in the network by the reading function and, for each such category C, a node such as M of Figure 6 is added. If no such categories exist, then only the initial string and the bounded string are added.

Forward inference may be triggered by the addition of the network of Figure 6 for each token, depending on what rules are already in the system. For example, in

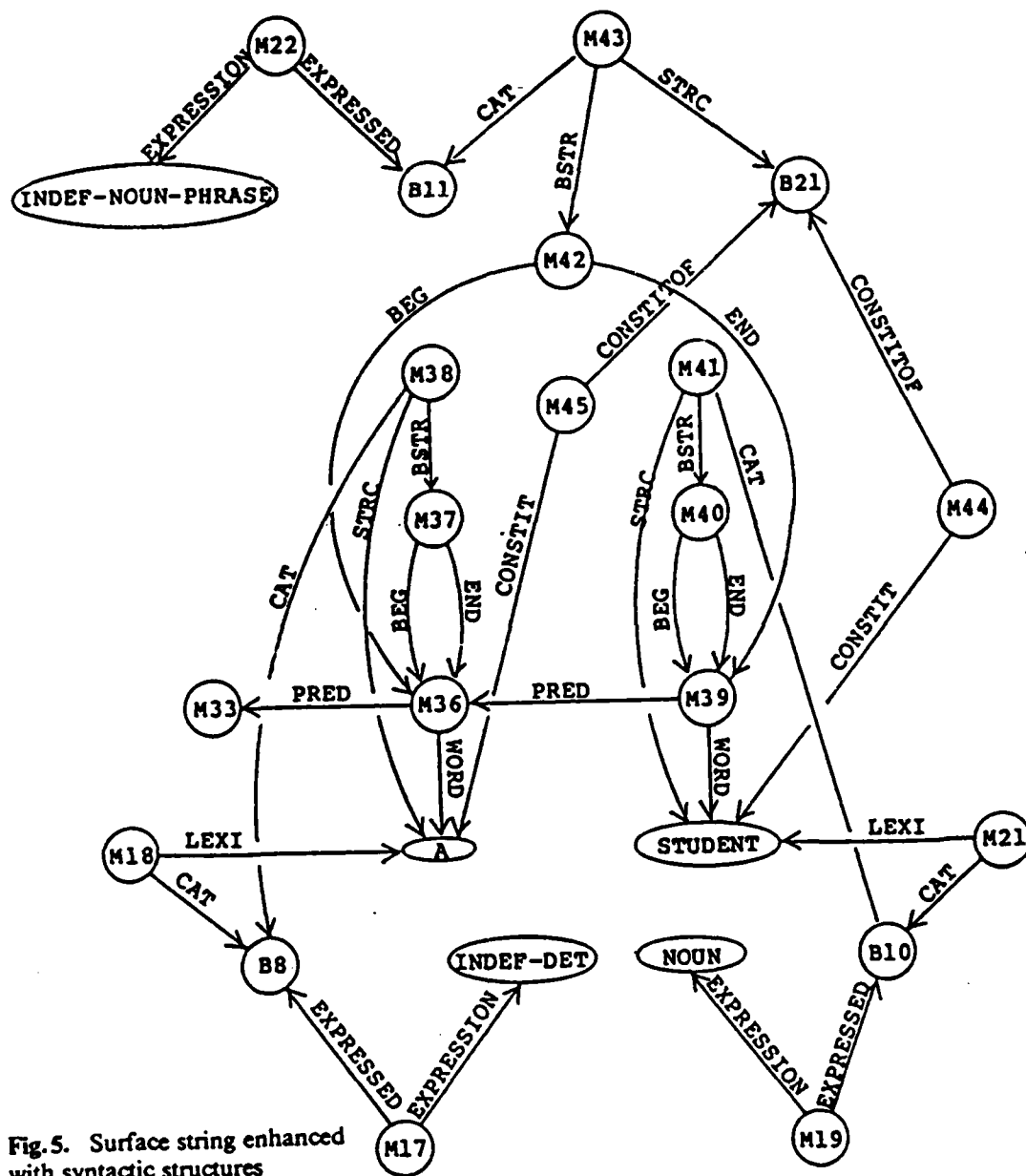


Fig. 5. Surface string enhanced with syntactic structures

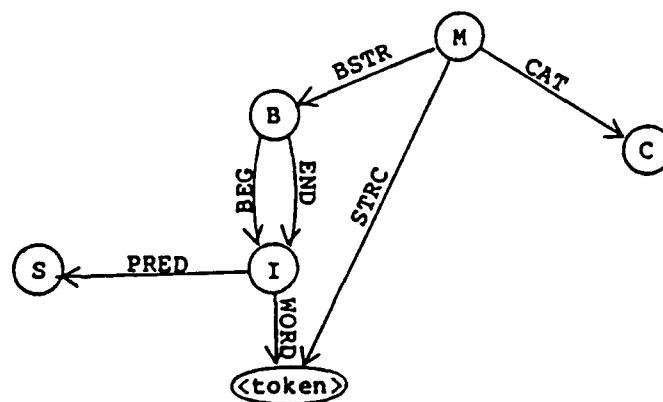


Fig. 6. The structure added to the network for each input token

Figure 5, nodes M38 und M41 are added by the reading function and nodes M42, M43, M44, and M45 are built only if there is a rule in the system that asserts that an INDEF-DET followed by a NOUN is an INDEF-NOUN-PHRASE.

2.4 The Representational Mapping

2.4.1 Introduction

Not all strings of a language form meaningful "chunks". For example, the substring "a large" from the sentence "A large aggressive dog frightened the girl" is not a conceptually coherent constituent of the sentence. Many researchers, e.g., Fodor and Garrett (1967), Bever (1970, 1973), and Levelt (1970, 1974), have investigated the relationship between surface constituents and the conceptually coherent components of an utterance. There seems to be good evidence for surface constituents being the coherent units for comprehension of discourse. How sentential constituents or discourse constituents (moving up to a higher level in the organization of text) are utilized in the comprehension process is an active field of research (Brown and Yule, 1983).

We let R designate the representational mapping (Allen, 1978) from surface strings to their interpretations. The domain of R contains the categories of strings that form conceptually coherent units, possibly depending on linguistic or other contexts. The domain of R initially contains predefined categories L-CAT, S-CAT, VARIABLE, LITERAL, LITERAL-STRING, VAR-APPOSITION-PHR, RULE-STMT, P-RULE, CASE-FRAME-DEFINITION, and CASE-SLOT-DEFINITION. They are discussed in the following sections.

We provide the teacher of the system with the facility for determining what the conceptually coherent constituents will be, in addition to the core, and for instructing the system in their use.

2.4.2 Base Cases

The categories L-CAT, S-CAT, VARIABLE, LITERAL, and LITERAL-STRING are the base cases for the representational mapping. The most basic subclass of the domain of R is L-CAT, the class of identifiers for the lexical categories of the system, including both system identifiers and user-defined identifiers. The class L-CAT contains the predefined identifiers L-CAT, S-CAT, and VARIABLE. The representational mapping applied to any identifier in L-CAT maps to a constant base node. In Figure 6, the interpretations of the identifiers L-CAT and S-CAT are represented by nodes B1 and B2 respectively. Similarly, if the system is informed that 'NOUN is an L-CAT, then its interpretation is represented by a base node (B4 of Fig. 7). The information that 'GOOSE is a NOUN and that 'NOUN-PHRASE is in S-CAT is also represented in Figure 7.

A member of VARIABLE maps to a corresponding network variable node. Since the interpretation of a user variable must be local to the rule in which it is used, the representational mapping applied to the class VARIABLE is handled in a special manner as explained in Section 3.

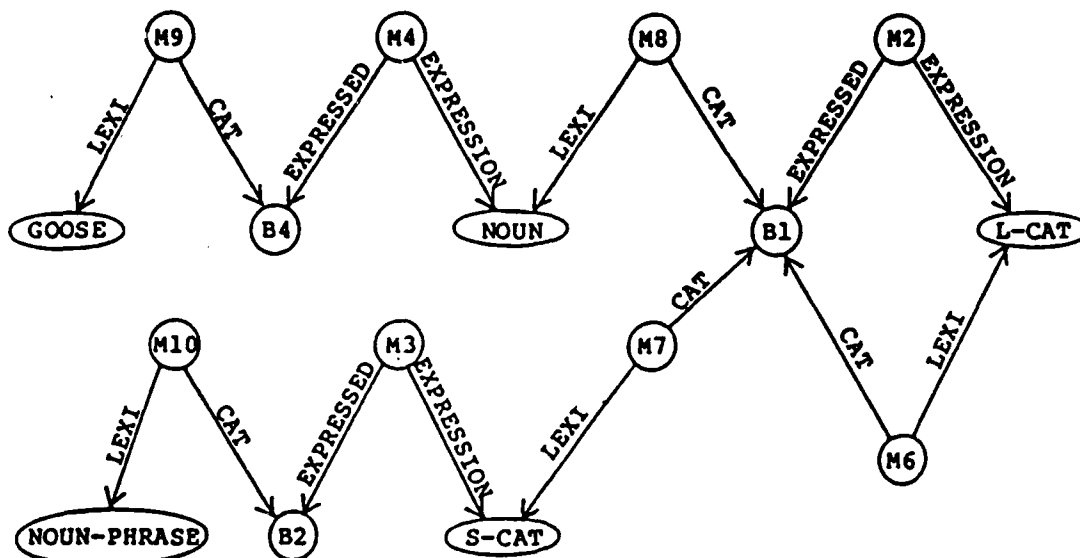


Fig. 7. Representation of some basic lexical knowledge

Fig. 8. The representational mapping applied to a LITERAL

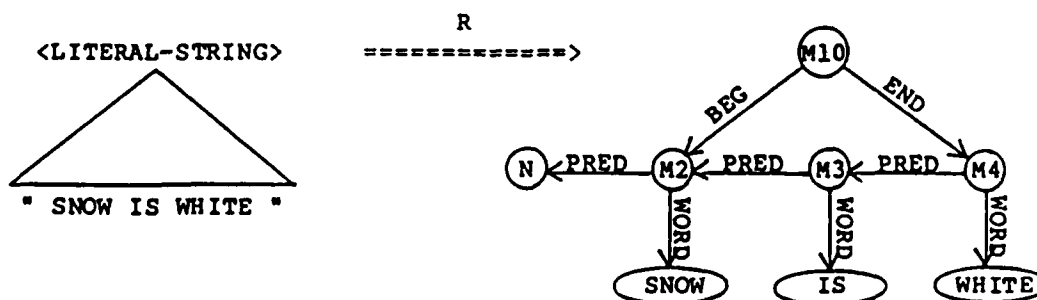
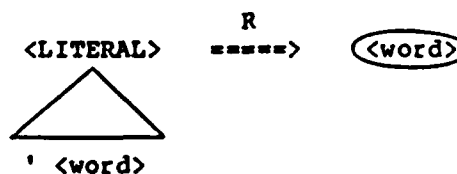


Fig. 9. Representational mapping applied to a LITERAL-STRING

The representational mapping applied to a LITERAL (defined in Section 2.2.1 as the single-quote mark followed by a word) maps to the node whose identifier is the word itself, as illustrated in Figure 8.

The representational mapping applied to a LITERAL-STRING (a string enclosed by double-quote marks) maps to the bounded string representing the string enclosed by the quote marks. Figure 9 illustrates the representational mapping applied to the literal string "SNOW IS WHITE".

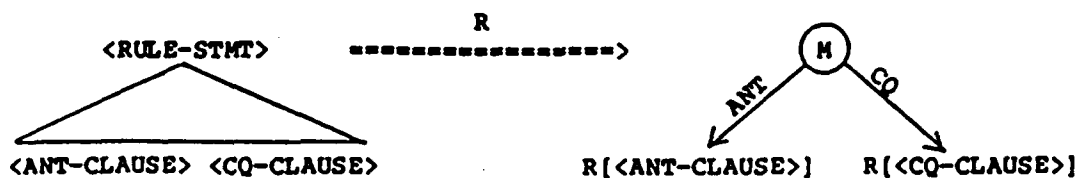


Fig. 10. Representational mapping applied to a RULE-STMT

2.4.3 Propositions and Structured Objects

Some string categories contained in the domain of R are mapped to non-atomic network (case frame) structures representing propositions or structured objects by the representational mapping. The system has just two predefined string categories, namely P-RULE and RULE-STMT, whose members' interpretations are represented as non-atomic structures by the representational mapping. P-RULEs have a predetermined syntax and are translated into SNePS network rules using the predefined structures. RULE-STMT is defined as the class of strings that are interpreted by the system as general rules. This class is initially empty and the syntax is to be determined by the teacher. A RULE-STMT must have an ANT-CLAUSE and a CQ-CLAUSE as constituents. The structure resulting from the application of the representational mapping R to a RULE-STMT is illustrated in Figure 10.

The ANT-CLAUSE category is defined as the class of strings that can be used in antecedent position in rules input by the teacher. Similarly, the CQ-CLAUSE category is defined as the class of strings that can be used in consequent position in rules input by the teacher. Both of these classes are initially empty and the syntax of RULE-STMTs, ANT-CLAUSES, and CQ-CLAUSES is to be determined by the teacher. An example is discussed in Section 3.

The teacher can add new string categories, whose interpretations are to be represented by non-atomic network structures, to the domain of R and specify the semantics of these string categories by using the semantic rewrite rule capability discussed in Sect. 2.5.3.

2.4.4 Participants in Propositions or Relations; Components of Structured Objects

In the previous section, the category of natural language phrases that assert relations between concepts or objects was discussed briefly. This type of phrase maps to the top node of a molecular representational structure.

Many phrases of natural language refer to individual concepts or objects that are participants in relations or propositions or that are components of structured objects. This type of phrase would map to a slot of one or more network case frame structures. The system has no predefined string categories which map to "participant" slots. The syntax for creating new categories of this type and their associated semantics is discussed in Section 2.5.3.

2.5 Kernel Language

2.5.1 Predefined Terms

As previously indicated, we are attempting to provide a facility with which a person can define a target language and yet keep the core as small and unbiased as possible. It is essential to provide the person (teacher) with a kernel language with which to start building up her language definition. The kernel language (KL) of our system consists of predefined terms, syntactic rewrite rules, and semantic rewrite rules. The predefined terms of the system are the names of the categories discussed in Section 2.2.1.

2.5.2 Syntactic Rewrite Rules

The kernel language includes linguistic rewrite rules to enable the teacher to instruct the system in the basic syntax of her target language.

a) **Lexical Entry:** The KL includes syntactic production rules of the form (L-CAT) → (LITERAL) where (L-CAT) represents the name of a lexical category that has already been defined. A LITERAL was defined in Section 2.2.1 as consisting of the single-quote followed by a word (the single-quote is part of the KL and indicates that the following word is mentioned rather than used). This form of production rule is the means of entering lexical items such as

L-CAT → 'NOUN
L-CAT → 'PROPER-NOUN
L-CAT → 'DEF-DET
L-CAT → 'INDEF-DET
L-CAT → 'VERB
L-CAT → 'BE-VERB
L-CAT → 'ADVERB
L-CAT → 'ADJECTIVE
NOUN → 'GOOSE
NOUN → 'GEESE
PROPER-NOUN → 'GRADY
PROPER-NOUN → 'GLADYS
DEF-DET → 'THE
INDEF-DET → 'A
VERB → 'HAS
BE-VERB → 'IS

ADVERB → 'THEN

L-CAT → 'PREPOSITION
L-CAT → 'CONJ
L-CAT → 'PROPERTY
S-CAT → 'HEAD-NOUN
S-CAT → 'STRING
VARIABLE → 'X
VARIABLE → 'Y

ADJECTIVE → 'WHITE
ADJECTIVE → 'SINGULAR
ADJECTIVE → 'PLURAL
PREPOSITION → 'OF
CONJ → 'IF
PROPERTY → 'COLOR
PROPERTY → 'NUMBER
UNIQUE-MEANING-CAT →
'ADJECTIVE
UNIQUE-MEANING-CAT →
'PROPERTY

b) Context Free Rules: The KL includes rules of the form

$\langle S-CAT \rangle \rightarrow \langle s \rangle_1 \dots \langle s \rangle_k, k > 0,$

where $\langle S-CAT \rangle$ represents the name of a string category and for each i , $\langle s \rangle_i$ is either a LITERAL, the name of a lexical category previously entered as a member of L-CAT as in (a) above, or the name of a string category.

Examples: PROPERTY-CLAUSE \rightarrow SUBJECT PREDICATE

SUBJECT \rightarrow NOUN-PHRASE

NOUN-PHRASE \rightarrow LITERAL

NOUN-PHRASE \rightarrow VARIABLE

NOUN-PHRASE \rightarrow PROPER-NOUN

PREDICATE \rightarrow RELATION-PREDICATE

PREDICATE \rightarrow BE-PREDICATE

RELATION-PREDICATE \rightarrow RELATION PREDICATE-ADJ

BE-PREDICATE \rightarrow BE-VERB PROPERTY-INDICATOR

RELATION \rightarrow 'HAS PROPERTY-INDICATOR

PROPERTY-INDICATOR \rightarrow PROPERTY-CLASS-INDICATOR

PROPERTY-INDICATOR \rightarrow PROPERTY

PROPERTY-CLASS-INDICATOR \rightarrow PREDICATE-ADJ

RULE-STMT \rightarrow 'IF ANT-CLAUSE 'THEN CQ-CLAUSE

c) Context Sensitive Rules: The KL includes syntactic production rules of the form

$\langle ls \rangle_1 \dots \langle ls \rangle_n \rightarrow \langle rs \rangle_1 \dots \langle rs \rangle_n, n > 0,$

where each element $\langle ls \rangle_i$ or $\langle rs \rangle_i$ is either a LITERAL, the name of a lexical category, or the name of a string category; both sides of the rule must have the same number of elements and for each element $\langle ls \rangle_i$ of the left side,

- 1) if $\langle ls \rangle_i$ is a LITERAL or lexical category, then the corresponding element $\langle rs \rangle_i$ of the right side must be the same as $\langle ls \rangle_i$;
- 2) if $\langle ls \rangle_i$ is the name of a string category, then the corresponding element $\langle rs \rangle_i$ of the right side can be either a LITERAL, lexical category name, or string category name.

This facility allows the user to enter context sensitive rules, such as:

RELATION PREDICATE-ADJ \rightarrow RELATION ADJECTIVE

RELATION PREDICATE-ADJ \rightarrow RELATION VARIABLE

BE-VERB PREDICATE-ADJ \rightarrow BE-VERB ADJECTIVE

'IF ANT-CLAUSE \rightarrow 'IF PROPERTY-CLAUSE

'THEN CQ-CLAUSE \rightarrow 'THEN PROPERTY-CLAUSE

The first rule asserts that in the context of a RELATION, an ADJECTIVE is recognized as a PREDICATE-ADJ, the second asserts that in the context of a RELATION, a VARIABLE is parsed as a PREDICATE-ADJ, and the third asserts that in the context of a BE-VERB, an ADJECTIVE is parsed as a PREDICATE-ADJ. Similarly, the fourth and fifth rules state that following the word "IF" or "THEN", a PROPERTY-CLAUSE is parsed as an ANT-CLAUSE or CQ-CLAUSE, respectively.

SUBJECT :: NOUN-PHRASE
 NOUN-PHRASE :: LITERAL
 NOUN-PHRASE :: VARIABLE
 NOUN-PHRASE :: PROPER-NOUN
 PROPERTY-INDICATOR :: PROPERTY-CLASS-INDICATOR
 PROPERTY-INDICATOR :: PROPERTY
 PREDICATE-ADJ :: ADJECTIVE
 PREDICATE-ADJ :: VARIABLE

define the semantics of a PROPERTY-CLAUSE to be a case frame with three slots, such that the PROPERTYOF slot is filled by the interpretation of the SUBJECT of the PROPERTY-CLAUSE, the PROPERTY slot is filled by the interpretation of the PROPERTY-INDICATOR constituent of the PROPERTY-CLAUSE, and the VALUE slot is filled by the interpretation of the PREDICATE-ADJ constituent. The second and third definitions above indicate that the interpretation of an ANT-CLAUSE is the same as the interpretation of its PROPERTY-CLAUSE constituent and that the interpretation of a CQ-CLAUSE is the same as the interpretation of its PROPERTY-CLAUSE constituent, respectively. The next rule defines the interpretation of a SUBJECT to be the same as the interpretation of its NOUN-PHRASE constituent. The next three rules define the interpretation of a NOUN-PHRASE to be the interpretation of either its LITERAL constituent, its VARIABLE constituent, or its PROPER-NOUN constituent, whichever it has. The remaining rules are similarly understood by the system.

The representational mapping R builds a network structure such as that dominated by node M of Figure 11 as the interpretation of a PROPERTY-CLAUSE.

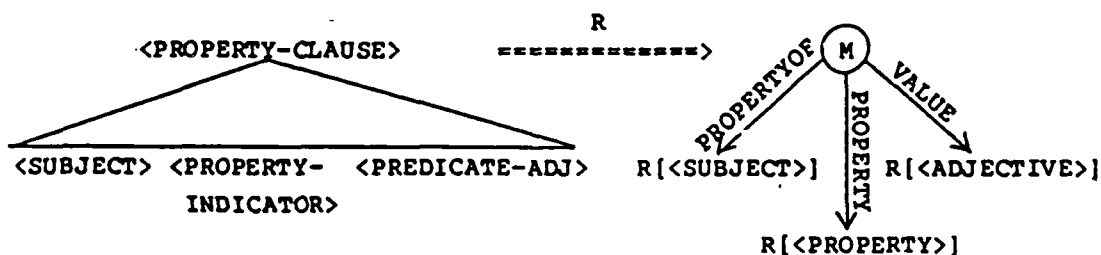


Fig. 11. Representational mapping applied to a PROPERTY-CLAUSE

b) Case Frame Slot-Filler Definitions: In order to provide a capability for defining the semantics of a phrase whose interpretation is a slot-filler in a case frame, the following type of semantic rewrite rule is included. The syntax of the CASE-SLOT-DEFINITION is

$$\langle \text{phr-name} \rangle > > ([(\langle \text{slot-name} \rangle_1 \langle \text{string-name} \rangle_1 \dots \langle \text{slot-name} \rangle_n \langle \text{string-name} \rangle_n)]^+$$

where the square brackets are part of the object language and the + and the parentheses are metasymbols. The <phr-name> is the name selected by the teacher for the category of strings whose semantics are defined by the expression to the right

2.5.3 Semantic Rewrite Rules

a) **Case Frame Definitions:** The KL includes language to enable the teacher to define case frames and instruct the system in their use by using the syntax of a CASE-FRAME-DEFINITION:

$$\langle \text{string-cat} \rangle :: \langle \text{slot-name} \rangle_1 \langle \text{constit-name} \rangle_1 \dots \langle \text{slot-name} \rangle_n \langle \text{constit-name} \rangle_n$$

where $n > 0$. Such a CASE-FRAME-DEFINITION is used by the system as follows: A string that is identified as being in category $\langle \text{string-cat} \rangle$ is mapped into a case frame such that for each slot identified by $\langle \text{slot-name} \rangle_i$, the slot-filler is the interpretation of the constituent string identified by $\langle \text{constit-name} \rangle_i$. The constituent strings need not be immediate constituents of the string in category $\langle \text{string-cat} \rangle$. The same $\langle \text{constit-name} \rangle$ can be used to specify the filler for more than one slot. For example, suppose the teacher wants to define a language in which an utterance such as "JOHN BOUGHT A HOUSE", involving the act of purchase, is interpreted to mean that the person bought the object for himself unless otherwise stated. To handle the semantics of such a clause, the teacher might want to define a case frame with AGENT and BENEFICIARY slots which are both filled by the interpretation of the same constituent of the clause. Our CASE-FRAME-DEFINITION facility provides for this eventuality.

If a string is parsed as a $\langle \text{string-cat} \rangle$ but is missing a constituent specified by the semantic rewrite rule, then a default representation of the slot-filler corresponding to the missing constituent is established in the form of an atomic node about which the system knows nothing, other than its being a participant in the case frame. In the context of a RULE-STMT the atomic node will be a variable node (see Section 1.3), otherwise a constant node. For example, to represent the interpretation of a sentence such as "THE HOUSE WAS PURCHASED YESTERDAY" the teacher might want to use the same case frame mentioned in the preceding paragraph. Since an AGENT and BENEFICIARY are implicitly part of the act of purchase, but not explicitly mentioned in the sentence, it is reasonable for the unmentioned participants to be represented in the interpretation of the sentence. The above default representation for the interpretation of a missing constituent provides the teacher with a facility for instructing the system how to interpret such a sentence.

An alternative syntax for the CASE-FRAME-DEFINITION is

$$\langle \text{string-cat} \rangle :: \langle \text{constit-name} \rangle.$$

The right side of the $::$ symbol is a degenerate case frame and the definition is interpreted as meaning that the semantics of the string of category $\langle \text{string-cat} \rangle$ is the same as that of the constituent string of category $\langle \text{constit-name} \rangle$. For example,

PROPERTY-CLAUSe	::	PROPERTY OF SUBJECT
		PROPERTY PROPERTY-INDICATOR
		VALUE PREDICATE-ADJ
ANT-CLAUSe	::	PROPERTY-CLAUSe
CQ-CLAUSe	::	PROPERTY-CLAUSe

of the symbol >>. The object language symbol + must be used in place of at least one (string-name), designating the position of the interpretation of the string (phr-name) in the case frame.

Each set of brackets encloses a case frame definition as described in the previous section. That is, each slot named (slot-name)_i is filled by the interpretation of a string in category (string-name)_i, if a string of category (string-name)_i is present as a (not necessarily immediate) constituent of the string of category (phr-name). The + symbol marks the slot whose filler is the interpretation of the (phr-name) string. The system represents the interpretation of the (phr-name) string (1) as a variable atomic node if the semantic rule is used in the context of a RULE-STMT and (2) as a constant atomic node, otherwise. If a slot-filler constituent is specified in a semantic rewrite rule, but is missing from the surface string to which the rule is being applied, a default representation of the slot-filler corresponding to the missing constituent is established in the form of an atomic node about which the system knows nothing, other than its being a participant in the case frame (as in the previous section for a CASE-FRAME-DEFINITION).

Consider the following example CASE-SLOT-DEFINITION:

PROPERTY-CLASS-INDICATOR >> [MEMBER ADJECTIVE PROPERTY-CLASS +]

According to this rule, a PROPERTY-CLASS-INDICATOR should have an ADJECTIVE constituent and the interpretation of a string of the PROPERTY-CLASS-INDICATOR category would be represented by an atomic node which fills the PROPERTY-CLASS slot of a case frame whose MEMBER slot is filled by the interpretation of the ADJECTIVE constituent. The mapping from a surface string to the network representation of its interpretation is illustrated in Figure 12.

In order to prepare for the example discussed in Sect.3, the following rule is input:

INDEF-S-PHRASE >> [BSTR STRING CAT S-CAT STRC +]

According to this rule, the interpretation of a string parsed by the system as an INDEF-S-PHRASE would be represented by an atomic node filling the STRC slot of a case frame whose CAT slot is filled by the interpretation of the S-CAT constituent of the INDEF-S-PHRASE and whose BSTR slot is filled by the STRING constituent. This is illustrated in Figure 13.

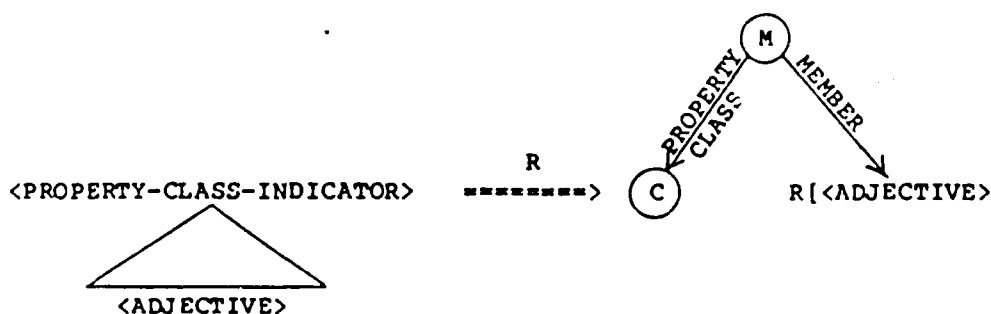


Fig. 12. Representational mapping applied to a PROPERTY-CLASS-INDICATOR

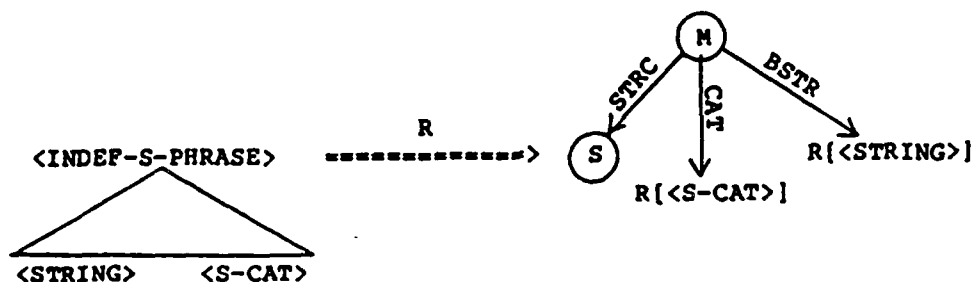


Fig. 13. Representational mapping applied to an INDEF-S-PHRASE

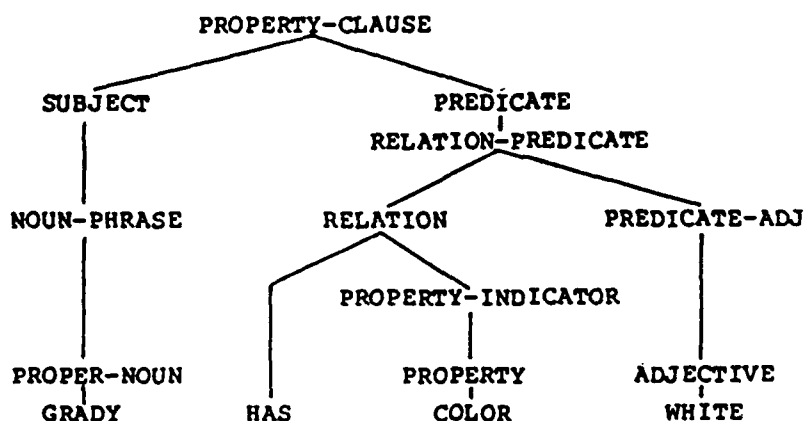


Fig. 14. Parse tree for sample input utterance

2.6 Use in Language Processing

To illustrate the system's use of the language definition developed via the rewrite rules of the preceding sections, we show some sentences of this language which refer to the language itself compared with some that refer to a non-linguistic domain. Thinking affectionately of her pet geese, the teacher informs the system that "GRADY HAS COLOR WHITE". The system recognizes and builds the parse tree of Figure 14 for the utterance. We show the more conventional form of the parse tree, rather than the equivalent network parse tree that the system actually builds in order to simplify the figure.

In the preceding sections, the teacher has entered rewrite rules into the system to define the semantics for certain string classes (e.g., PROPERTY-CLAUSE, PROPERTY-INDICATOR), thereby identifying the conceptually coherent constituents for the language definition. The system applies these semantic rewrite rules and builds the structure of Figure 15 as the interpretation of the utterance. The assertion that the parsed input utterance expresses the concept represented by node M75 is also established in the network.

Similarly, the system can process the input utterance "'GOOSE HAS NUMBER SINGULAR". The resulting parse tree is shown in Figure 16.

The representation of the interpretation of the utterance is shown in Figure 17.

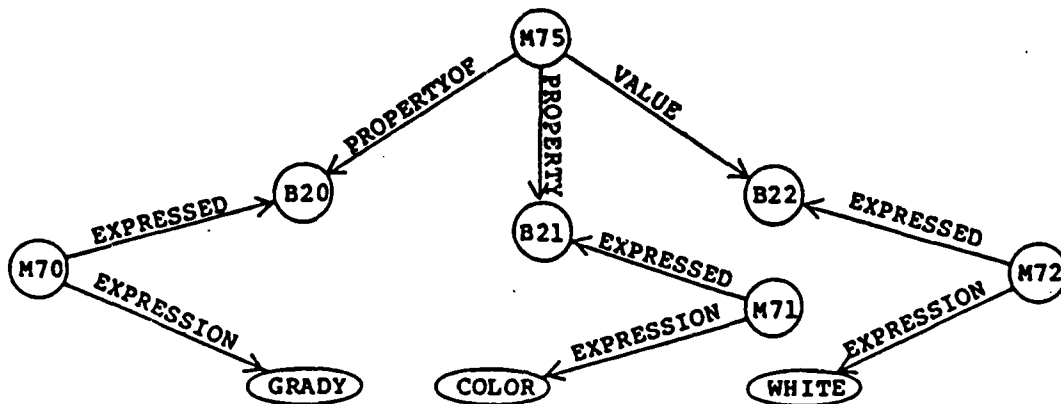


Fig. 15. Representation of the interpretation of input utterance

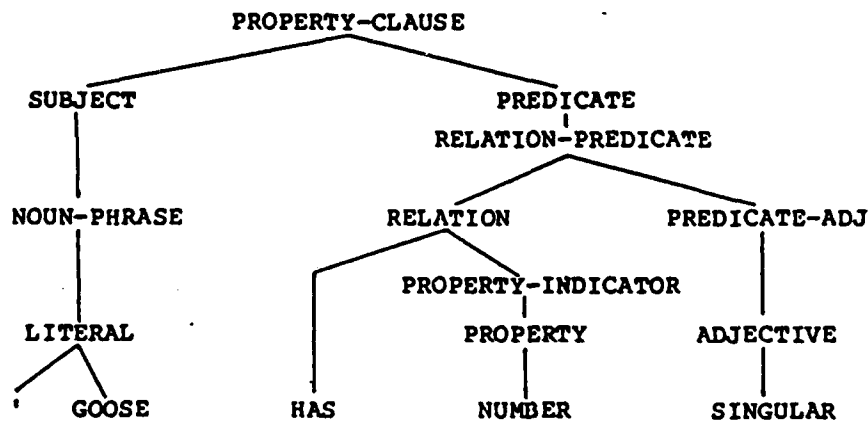


Fig. 16. Parse tree for utterance concerning language

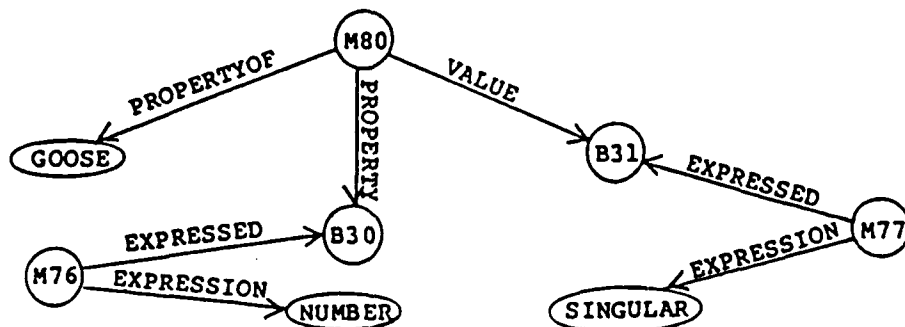


Fig. 17. Representation of interpretation of utterance

As stated in Section 1.5, the NL system distinguishes between a word or phrase and its interpretation. The interpretation of a LITERAL is the word following the quote mark (the word 'GOOSE', in this case). Thus node M80 represents the proposition that singular number is a property of the word 'GOOSE' and not of the concept expressed by the word 'GOOSE'. On the other hand, the interpretation of the word 'GRADY' is represented by node B20 of Figure 15, and it is this entity

that has color white. Comparing these two examples illustrates the knowledge representations we have established as well as the capability for handling strings and their interpretations as domain knowledge, which is fundamental to our theory and system.

At this stage, the teacher can simplify the language to use with the system for expressing properties. She does this by inputting the following rewrite rules so that property class entries can be made.

PROPERTY-CLASS-ENTRY → ADJECTIVE 'IS 'A PROPERTY
 PROPERTY-CLASS-ENTRY :: MEMBER ADJECTIVE
 PROPERTY-CLASS PROPERTY

Since the system has previously been informed that 'WHITE is an ADJECTIVE and 'COLOR is a PROPERTY, the utterance "WHITE IS A COLOR" would be recognized by the system as a PROPERTY-CLASS-ENTRY. Also since, in Section 2.5.3, for the purposes of this example, the teacher entered 'ADJECTIVE and 'PROPERTY into UNIQUE-MEANING-CAT, the surface strings that are in the categories ADJECTIVE and PROPERTY are each treated as having a unique interpretation. Thus different instances of the same string such as 'WHITE are treated by the system as having the same interpretation and it uses just one network structure to represent this interpretation. Therefore, using the above semantic rewrite rule, the system builds the structure of node M85 of Figure 18 to represent the interpretation of the utterance, finding the nodes B21 and B22 of Figure 15 to represent the interpretation of 'COLOR and 'WHITE, respectively.

Similarly, the system can be informed that "PLURAL IS A NUMBER" and it builds a structure similar to that of Figure 18 to represent the assertion that the concept expressed by 'PLURAL is a member of the property-class NUMBER.

If the utterance "GLADYS IS WHITE" is now input to the system, the utterance is also recognized as a PROPERTY-CLAUSE as shown in Figure 19.

The semantic rewrite rules of the previous section are used by the system to build the structure dominated by node M90 as the representation of the interpretation of the utterance.

According to the semantic rule for a PROPERTY-CLAUSE, the PROPERTY slot in the case frame is filled by the interpretation of the PROPERTY-INDICATOR constituent of the utterance. Referring to the parse tree of Figure 19, the PROPERTY-INDICATOR consists of the PROPERTY-CLASS-INDICATOR. The semantic rule

PROPERTY-INDICATOR :: PROPERTY-CLASS-INDICATOR

of the previous section instructs the system to use the interpretation of the PROPERTY-CLASS-INDICATOR as the interpretation of the PROPERTY-INDICATOR. The rule for interpreting a PROPERTY-CLASS-INDICATOR is the CASE-SLOT-DEFINITION presented in the previous section:

PROPERTY-CLASS-INDICATOR > > [MEMBER ADJECTIVE PROPERTY-CLASS +]

This rule instructs the system to interpret the PROPERTY-CLASS-INDICATOR as the PROPERTY-CLASS slot-filler of the frame whose MEMBER SLOT is

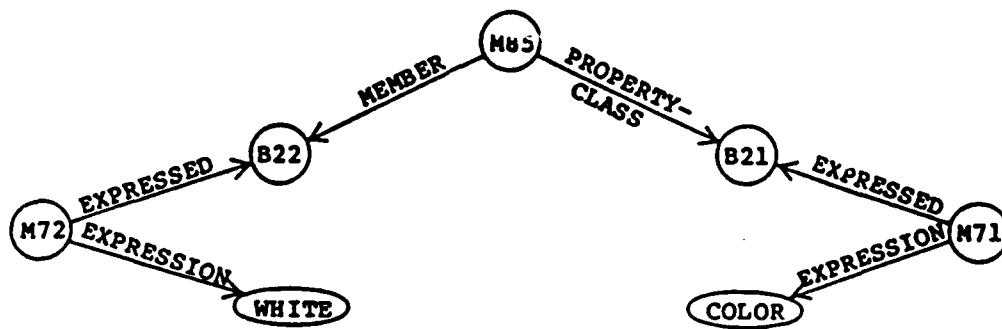


Fig. 18. Representation of interpretation of utterance

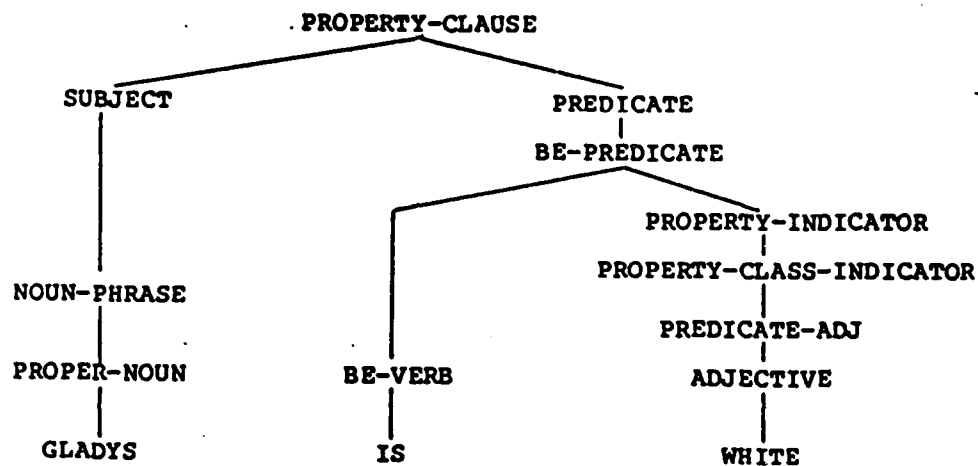


Fig. 19. Parse tree for input utterance

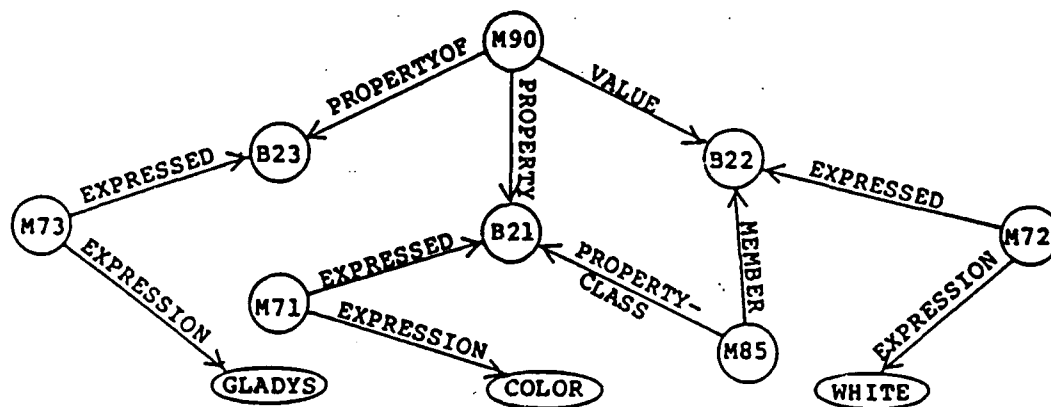


Fig. 20. Representation of interpretation of utterance

filled by WHITE. Node B22 of Figures 15 and 18 is found as the representation of the interpretation of 'WHITE', since the members of the class ADJECTIVE have been defined by the teacher as having "unique semantics". Thus, the system uses node B22 to use as the MEMBER slot-filler for the case frame associated with a PROPERTY-CLASS-INDICATOR. Then B21 of Figure 15 is found and used as the PROPERTY-CLASS slot-filler as shown in Figure 20, since it represents the

PROPERTY-CLASS that has WHITE as a MEMBER. Node B21 is also the representation of the interpretation of the PROPERTY-CLASS-INDICATOR string. In general, a CASE-SLOT-DEFINITION maps a surface string to a participant in a relation or proposition.

In a manner similar to the parsing and interpretation of the utterance "GLADYS IS WHITE", the system also understands the utterance "'GEESE IS PLURAL". The system's language definition is again used as a metalanguage to expand upon the same language itself.

3 Increasing the System's Language Capability Through Its Language Capability

3.1 Motivation

Since we treat linguistic knowledge as domain knowledge, the system teacher (user) can add to the knowledge base and instruct the system as to how to process or understand ever more sophisticated language.

Just as a person is continually influenced by interaction with his environment, the data base of our system is modified by each input. The knowledge base is incrementally enhanced to form a more sophisticated system.

Since we represent language processing knowledge in the same knowledge base and in the same formalism as other domain knowledge, it is possible to make the system's language processing knowledge the subject of its language processing and this is a fundamental aspect of our approach. Thus by instructing the system in the domain of linguistics as we would expect to be able to do with another domain in an interactive NLU system, we can increase the system's language capability through its language capability. A user can communicate with our system in just one language via one processor without switching "modes" or interacting with supportive processors in special purpose languages.

The rewrite rules of the KL are certainly not sufficient for expressing all the rules a teacher would need to define a language of her choice. Therefore, one of the most important capabilities that the system needs is to understand a more general form of rule statement. A teacher should be able to bootstrap into a more powerful rule statement language from the KL. In the next sections, we present an example from such a bootstrap process.

3.2 Defining More-General Rule Forms

The teacher first extends the system's language definition so that it can begin to understand general "IF-THEN" rules. As stated in Section 2.2.1, RULE-STMT is a predefined category. The syntax of a RULE-STMT is not predefined, but for the interpretation process, each RULE-STMT must have an ANT-CLAUSE and a CQ-CLAUSE constituent. The ANT-CLAUSE constituent is interpreted as the antecedent of the rule and the CQ-CLAUSE constituent as the consequent of the rule. Thus the rewrite rule

RULE-STMT → 'IF ANT-CLAUSE 'THEN CQ-CLAUSE

defines a syntax for the RULE-STMT. The syntax and semantics of ANT-CLAUSE and CQ-CLAUSE must also be defined. This was done in Section 2.5 (see Appendix).

The additional rules that the teacher chooses to input to the system to increase its capability of understanding linguistic-domain language for this example are listed below. In order to make use of the system's ability to use a VARIABLE as an appositive to another phrase and remember the association of the VARIABLE to the phrase, the teacher inputs:

DEF-S-PHRASE → DEF-DET S-CAT
INDEF-S-PHRASE → INDEF-DET S-CAT
MAIN-APPOS-PHR VAR-NAME → INDEF-S-PHRASE VARIABLE
VAR-APPOSITION-PHR → MAIN-APPOS-PHR VAR-NAME

To explain to the system how to parse and interpret language which describes one phrase being a constituent of another, the teacher inputs:

SUP-STRING-REF → VAR-APPOSITION-PHR
CONSTIT-REF → DEF-S-PHRASE
CONSTIT-PHRASE → CONSTIT-REF 'OF SUP-STRING-REF
NOUN-PHRASE → CONSTIT-PHRASE
CONSTIT-PHRASE > > [CONSTIT + CONSTIT OF SUP-STRING-REF]
[BSTR STRING CAT DEF-S-PHRASE STRC +]
SUP-STRING-REF :: VAR-APPOSITION-PHR
MAIN-APPOS-PHR :: INDEF-S-PHRASE
NOUN-PHRASE :: CONSTIT-PHRASE
DEF-S-PHRASE :: S-CAT

These rules will be used in the next sections.

3.3 Parsing Strategy

The parsing strategy applied by our NL system is a combined bottom-up, top-down strategy. As each word of an input string is read by the system, the network representation of the string is extended as discussed in Section 2.3 and relevant rules stored in the SNePS network are triggered. All applicable rules are started in parallel in the form of processes created by our MULTI-processing package (McKay and Shapiro, 1980). These processes are suspended if not all their antecedents are satisfied and are resumed if more antecedents are satisfied as the reading of the string proceeds. As parsing proceeds, the annotated parse (tree(s) for an input utterance is (are) represented in the system's network knowledge base. Our system builds and retains network structures corresponding to alternative analyses of a given input string. Retention of the alternatives avoids the reanalysis of previously processed surface strings that occurs in a backtracking system.

Processing is controlled by the SNePS Inference Package (Shapiro et al., 1982), which employs bi-directional inference. This is a form of inference resulting from

interaction between forward and backward inference and loosely corresponds to bi-directional search through a space of inference rules. This technique focuses attention towards the active parsing processes and prunes the search through the space of inference rules by ignoring rules which have not been activated. This cuts down the fan out of pure forward or backward chaining. New rules are activated only if no active rules are applicable.

Consider the sample input utterance "IF THE HEAD-NOUN OF A NOUN-PHRASE X HAS NUMBER Y THEN X HAS NUMBER Y". When the first word is read by the system, it is recognized as matching the word 'IF in the rules

RULE-STMT \rightarrow 'IF ANT-CLAUSE THEN CQ-CLAUSE
'IF ANT-CLAUSE \rightarrow 'IF PROPERTY-CLAUSE

and parsing begins in a bottom-up manner. Both rules are triggered in parallel by the SNePS MULTI package. When originally input, each of the above rules was interpreted by the system and stored in the form of a network rule which we paraphrase as follows (NOTE: In all the paraphrased rules of this section, V_1 and V_2 are universally quantified variables):

- (1) If a word of an input string is the word 'IF, then
- (2) if V_1 follows the word 'IF and V_1 is an ANT-CLAUSE, then
- (3) if the word 'THEN follows V_1 , then
- (4) if V_2 follows the word 'THEN and V_2 is a CQ-CLAUSE,
then the string consisting of 'IF followed by V_1
followed by 'THEN followed by V_2 is a RULE-STMT.
- (5) If a word of an input string is the word 'IF, then
- (6) if V_1 follows the word 'IF and V_1 is a PROPERTY-CLAUSE,
then V_1 is an ANT-CLAUSE.

(The numbers in parentheses are rule numbers, not line numbers. Thus, for example, nested rule (3) begins with "if the word 'THEN'" and continues to the period at the end of the sentence.)

Since the antecedent of rule (1) above is satisfied, the system questions whether a string immediately following the word 'IF is an ANT-CLAUSE. When a SNePS rule is triggered, a process is created forming the active version of the rule for the purpose of such activities as data collection and variable binding. Some of these processes act as demons, waiting for instances of their antecedents so that instances of their consequents can be deduced. This is the case for the nested rule (2). Since no string follows the word 'IF yet, the process for rule (2) is suspended.

These active processes, with their communication links, form the equivalent of a hypothesized parse tree with associated expectations. The inference system ignores unactivated rules as long as there are applicable active rule processes awaiting data, essentially parsing in a top-down manner in this situation. The hypothesized parse tree corresponding to the process of rule (2) is illustrated in Figure 21. The parsing strategy of our system is similar to "left-corner bottom-up parsing" (Burge, 1975) in that construction of a parse tree begins at the bottom left corner, processing of a surface string proceeds in a left-to-right manner, and whenever an initial segment of a string has been parsed, the system attempts to establish a goal analysis of the string or substring thereof. In the following figures, the bro-

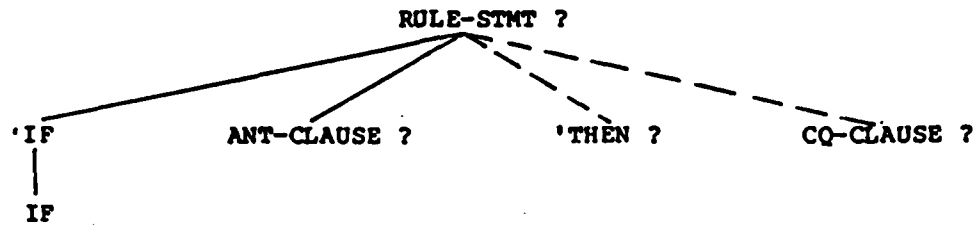


Fig. 21. Hypothesized parse tree

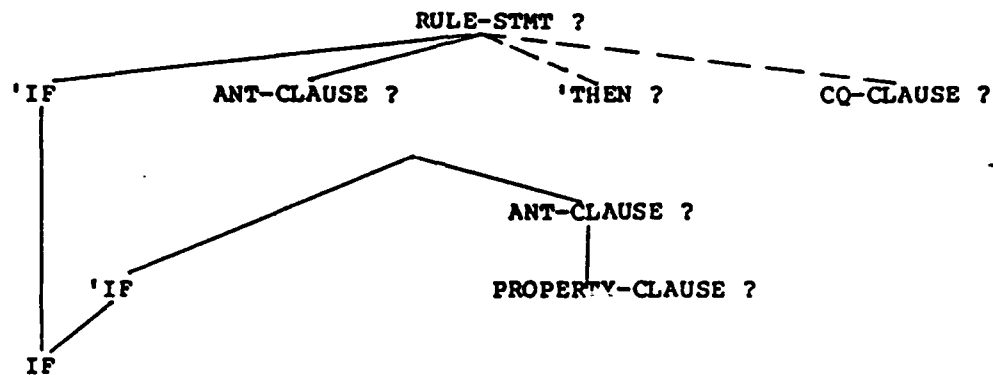


Fig. 22. Hypothesized parse tree

ken lines indicate goals or expectations represented by antecedents of nested rules for which active demons have not yet been created. The question-marks indicate expectations which have not yet been satisfied.

The antecedent of rule (5) is also satisfied. This is a context sensitive rule which constrains the parsing process. According to this rule, a PROPERTY-CLAUSE is parsed as an ANT-CLAUSE in the context of the word 'IF. A process is created forming the active version of rule (6) and this process awaits a PROPERTY-CLAUSE following the word 'IF. Figure 22 reflects the current state of the system in terms of its active processes, implicit expectations, and the tokens that it has consumed.

When the word 'THE is read by the system, the rule

DEF-S-PHRASE \rightarrow DEF-DET S-CAT

is triggered as parsing continues in a bottom-up manner. This rule is paraphrased as:

- (7) If V_1 is a DEF-DET, then
- (8) if V_2 follows V_1 and V_2 is an S-CAT,
then the string consisting of V_1 followed by V_2 is a
DEF-S-PHRASE.

The antecedent of rule (7) is satisfied and a process is created for nested rule (8) to await an S-CAT following the DEF-DET. The active processes form another hypothesized parse tree shown in Figure 23.

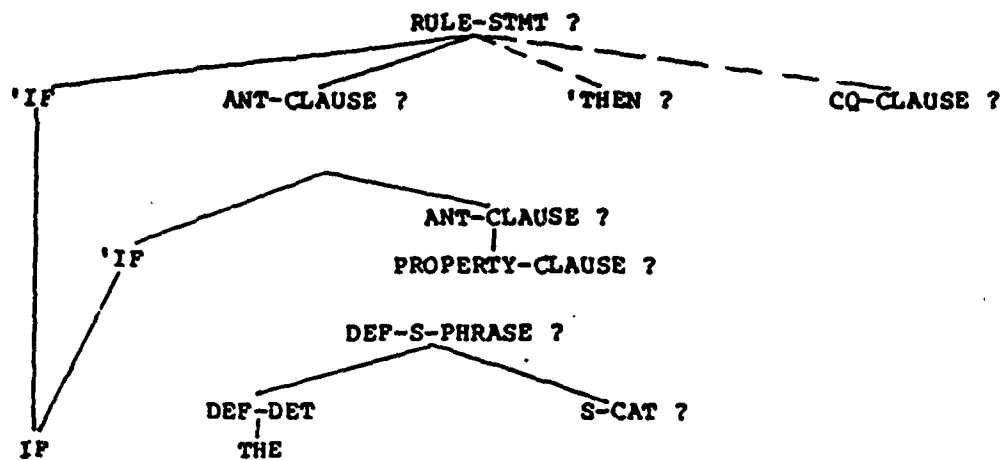


Fig.23. Hypothesized parse tree

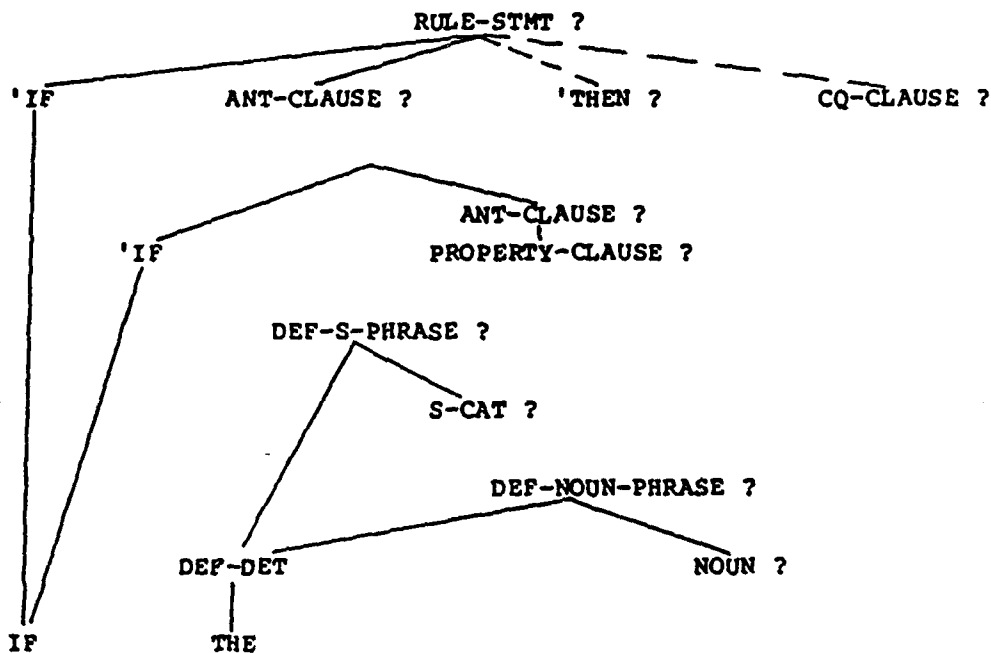


Fig.24. Hypothesized parse trees

Suppose another rule such as $\text{DEF-NOUN-PHRASE} \rightarrow \text{DEF-DET NOUN}$ had been entered by the teacher and is present in the network knowledge base. This rule is paraphrased as:

- (9) If V_1 is a DEF-DET, then
- (10) if V_2 follows V_1 and V_2 is a NOUN, then the string consisting of V_1 followed by V_2 is a DEF-NOUN-PHRASE.

This latter rule is also triggered by the system's reading of the word 'THE' and the processes created for rules (9) and (10) form another set of hypothesized parse trees as illustrated in Figure 24.

The parse trees of Figure 24 dominated by DEF-S-PHRASE? and DEF-NOUN-PHRASE? represent alternative possibilities for the parse of the string beginning with the word 'THE. A process such as the process for rule (10) waiting for a NOUN may remain suspended indefinitely if the expected data is not forthcoming.

When the next word 'HEAD-NOUN is read, the system recognizes it as an S-CAT and the process corresponding to rule (8) is resumed since it is waiting for an S-CAT following the word 'THE. Thus the string "THE HEAD-NOUN" is recognized as a DEF-S-PHRASE by application of the teacher's rules. This DEF-S-PHRASE then triggers the network version of the following rule and the DEF-S-PHRASE is then recognized as a CONSTIT-REF:

CONSTIT-REF \rightarrow DEF-S-PHRASE

Recognition of a CONSTIT-REF triggers the rule

CONSTIT-PHRASE \rightarrow CONSTIT-REF 'OF SUP-STRING-REF

whose network representation can be paraphrased as

- (11) If V_1 is a CONSTIT-REF, then
- (12) if the word 'OF follows V_1 , then
- (13) if V_2 follows the word 'OF and V_2 is a SUP-STRING-REF,
then the string consisting of V_1 followed by the word
'OF followed by V_2 is a CONSTIT-PHRASE.

Activation of rule (11) is analogous to bottom-up processing again. A process is established for rule (12) to await the word 'OF in the input stream.

When the next word 'OF is read by the system, the demon corresponding to rule (12) is activated and since the antecedent of rule (12) is satisfied, a process is established for rule (13) to expect a SUP-STRING-REF following the word 'OF. No other rules are activated by the reading of the word 'OF since an active process was waiting for this word in the input stream.

The system parses the next string "A NOUN-PHRASE" as an INDEF-S-PHRASE by application of the rule

INDEF-S-PHRASE \rightarrow INDEF-DET S-CAT

This triggers the rule

MAIN-APPOS-PHR VAR-NAME \rightarrow INDEF-S-PHRASE VARIABLE

which is paraphrased as

- (14) If V_1 is an INDEF-S-PHRASE, then
- (15) if V_2 follows V_1 and V_2 is a VARIABLE,
- (16) then V_1 is a MAIN-APPOS-PHR and V_2 is a VAR-NAME.

Since the antecedent of rule (14) is satisfied, a process is set up for rule (15). When the next word 'X is read, it is recognized as a VARIABLE and since the active process for rule (15) is waiting for a VARIABLE, no unactivated rules are applied. An example of such an unactivated rule is

NOUN-PHRASE → VARIABLE

which we previously input to the system. Thus an alternative parse is blocked by the expectation of a VARIABLE by the process for rule (15). By application of the rules

VAR-APPOSITION-PHR → MAIN-APPOS-PHR VAR-NAME

SUP-STRING-REF → VAR-APPOSITION-PHR

the expected SUP-STRING-REF of rule (13) is satisfied and the string "THE HEAD-NOUN OF A NOUN-PHRASE X" is parsed as a CONSTIT-PHRASE. By application of the rule

NOUN-PHRASE → CONSTIT-PHRASE

the string is also recognized as a NOUN-PHRASE. Notice that the term NOUN-PHRASE is *mentioned* in the input string and *used* in the application of the above rule.

At this point in the parsing process the hypothesized parse trees are illustrated in Figure 25.

As parsing proceeds using the rules introduced in this and preceding sections of this chapter, the resulting parse of the entire input statement is shown in Figure 26. The string category identifiers in the tree that are underlined are the categories that are included in the domain of the representational mapping. These are the categories for which the teacher has defined a rule to determine the interpretation of any member of the category (i.e., the underlining identifies the string categories defined by the teacher as the conceptually coherent constituents of the utterance).

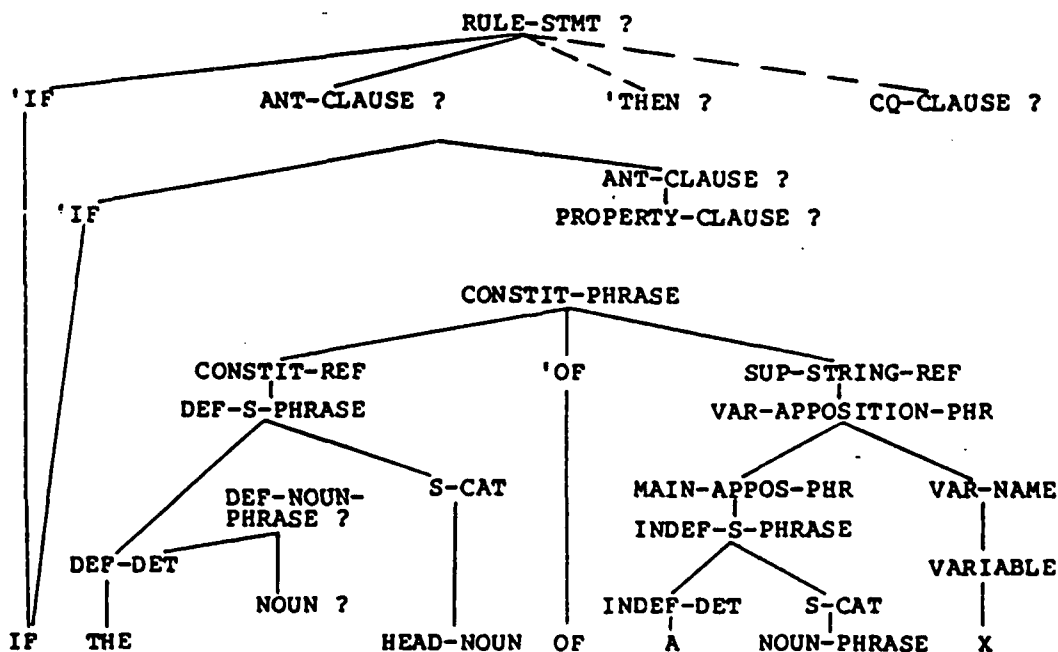


Fig. 25. Hypothesized parse trees

In this section on parsing, we have illustrated the following characteristics of our system's strategy:

- (1) the parallel processing of applicable rules;
- (2) constraint of the parsing process by the use of context sensitive rules;
- (3) constraint of the parsing process by the SNePS Inference Package focusing on active rule processes - the manifestation being the blocking of multiple parses by previously established expectations;
- (4) suspension and resumption of rule processes during the parsing process.

The retention of alternative analyses of a string, which avoids the reanalysis of certain strings in the case of a backtracking system, was not illustrated by the example of this section, but is a characteristic of our system.

Also of importance in this section is the fact that the system is again using its acquired language definition as a metalanguage to understand another instruction from the teacher concerning the language itself.

3.4 Interpretation of the Input Rule Statement

During the interpretation process, a VARIABLE of the user's language is translated into a variable node of the semantic network. The scope of a user VARIABLE is the utterance in which it occurs. The association of a user VARIABLE to its interpretation is maintained on a list only during translation of the utterance in which the VARIABLE occurs.

The interpretation of a user VARIABLE is as follows: If a VARIABLE is used as the VAR-NAME of a VAR-APPOSITION-PHR, discussed briefly in Section 2.2.1, then the system uses the interpretation of the MAIN-APPOS-PHR as the interpretation of the VARIABLE, and stores this association on the variable association list. Otherwise, the system checks the variable association list for a corresponding interpretation already established. Otherwise, a new variable node is created as the interpretation of the user VARIABLE, the new pair once again being added to the variable association list.

As shown in Figure 25, the phrase "A NOUN-PHRASE X" was recognized by the system as a VAR-APPOSITION-PHR, with "A NOUN-PHRASE" recognized as the MAIN-APPOS-PHR and 'X' as the VAR-NAME. Thus the interpretation of the phrase "A NOUN-PHRASE" is remembered by the system as the interpretation of 'X'. The string "A NOUN-PHRASE" has been recognized as an INDEF-S-PHRASE and thus the semantic rule

INDEF-S-PHRASE > > [BSTR STRING S-CAT STRC +]

of Section 2.5.3 applies. As discussed in Section 2.5.3, if a constituent is mentioned in a semantic rule but is missing from the surface string to which the rule applies, then the system represents the interpretation of the constituent as an atomic node. Furthermore, this atomic node is a variable node in the context of a RULE-STMT. Since the slot-filler constituent of category STRING is not present in our example INDEF-S-PHRASE, an atomic variable node (V2 of Fig. 26) is built to represent

the interpretation of the missing STRING constituent. The representation of the interpretation of the S-CAT constituent "NOUN-PHRASE" is node B25, representing the category of NOUN-PHRASEs. The STRC slot-filler becomes the interpretation of the INDEF-S-PHRASE. This slot-filler is also represented by an atomic variable node (V1 of Fig. 26) as explained in Sect. 2.5.3. The + symbol in the rewrite rule marks the participant of the proposition represented by the case frame whose representation is also the representation of the interpretation of the INDEF-S-PHRASE. Thus the interpretation of the INDEF-S-PHRASE "A NOUN-PHRASE" is node V1 of Figure 26. That is, the INDEF-S-PHRASE is interpreted as a variable node to be instantiated by a structure representing an analyzed surface string which has an associated bounded-string (see Sect. 2.2.3) and category NOUN-PHRASE. V1 is also the interpretation of user VARIABLE 'X' due to the string "A NOUN-PHRASE X" being a VAR-APPOSITION-PHR and the association of V1 and 'X' is stored on the variable association list.

The input string "THE HEAD-NOUN OF A NOUN-PHRASE X" was parsed as a CONSTIT-PHRASE (refer to Fig. 26). The rule for interpreting a CONSTIT-PHRASE was given in Section 3.3 as

```

CONSTIT-PHRASE >> [CONSTIT + CONSTITOF SUP-STRING-
                    REF]
                    [BSTR STRING CAT DEF-S-PHRASE STRC
                    +]

```

This rule stipulates that the interpretation of a CONSTIT-PHRASE is a participant in two case frames as defined in the two sets of brackets and the + symbol marks the slot-filler that is the interpretation of the CONSTIT-PHRASE. Again an atomic variable node is built to represent this slot-filler which also represents the CONSTIT-PHRASE. The SUP-STRING-REF is the constituent "A NOUN-PHRASE X" (refer to Fig. 26), whose interpretation is represented by node V1 of Figure 27. The structure representing the case frame defined in the second set of brackets is built in a manner similar to that used in building the structure of Figure 27 and described above.

The interpretation of the example CONSTIT-PHRASE "THE HEAD-NOUN OF A NOUN-PHRASE X" is represented by node V3 of Figure 28. The node V1 of Figure 28 is the same node as V1 of Figure 27.

Assembling the interpretations of the constituents of our RULE-STMT from Figures 27 and 28 and completing the interpretation of the RULE-STMT as the system does using the semantic rewrite rules of this article, node M86 of Figure 29 represents the interpretation of the RULE-STMT. All of the variable nodes V1, V2,

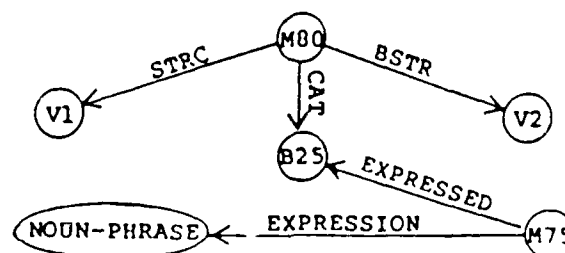


Fig. 27. Node V1 represents the interpretation of "A NOUN-PHRASE"

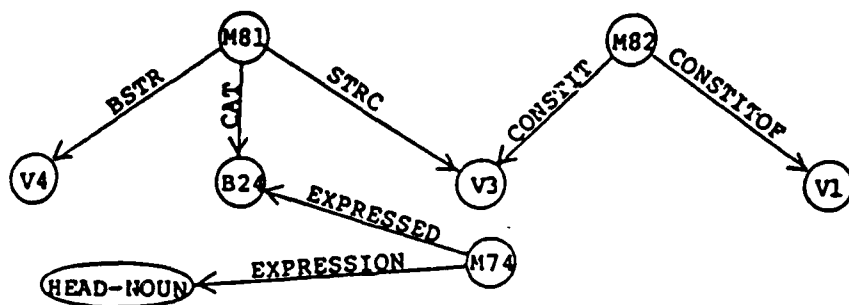


Fig. 28. Node V3 represents the interpretation of the CONSTIT-PHRASE

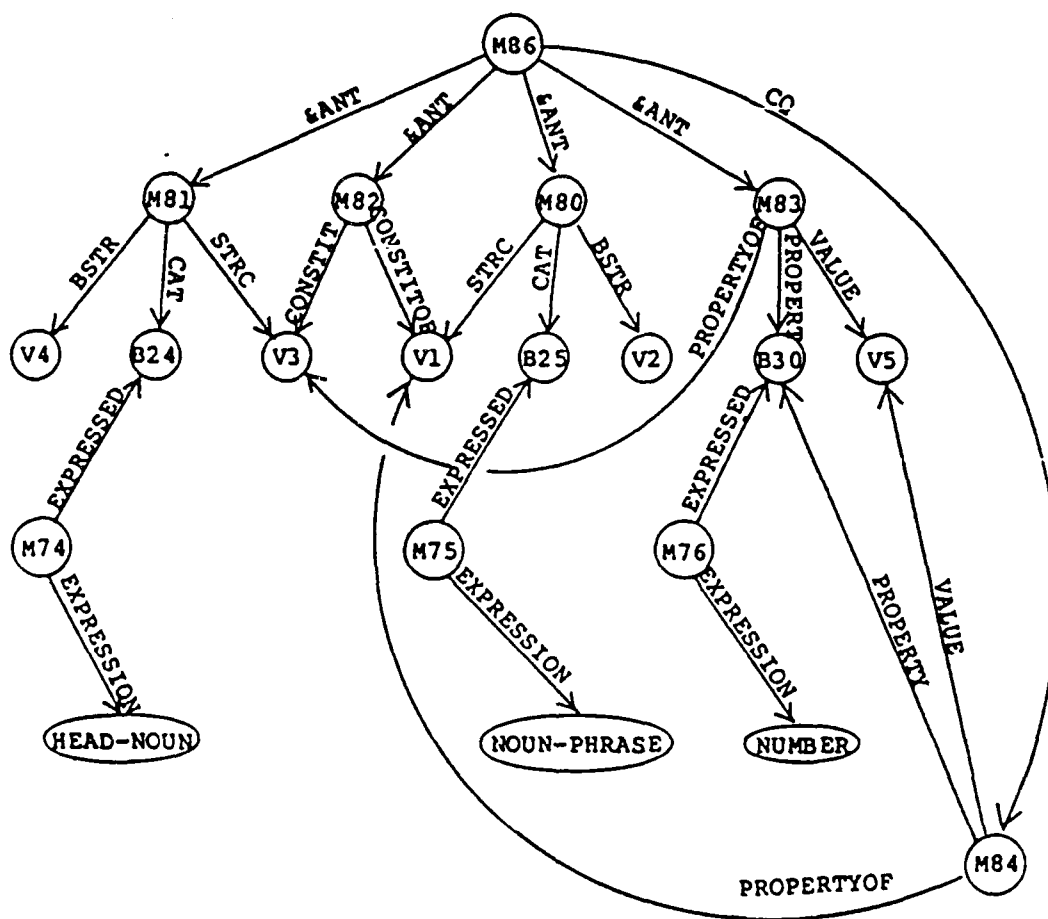


Fig. 29. Node M86 represents the interpretation of the input rule

V3, V4, and V5 are universally quantified (refer to Shapiro (1979 a) for the network representation of the quantification that is not shown in the figure and for more details on the rule structures of SNePS). The &ANT and CQ arcs are the SNePS system arcs used in the network representation of "&-entailment", the entailment of any of a set of consequents by the conjunction of one or more antecedents.

4 Language Use-Mention Distinction

In order for our system to treat linguistic knowledge as domain knowledge and to receive instruction in the use of this knowledge, it is essential for the system to distinguish between use and mention of language (Quine, 1951). We have already seen examples of this capability in our system. When words are entered into their appropriate lexical categories as in Section 2.5.2a, they are *mentioned*. Those lexemes that are themselves names of lexical categories are subsequently *used* to refer to their corresponding lexical categories. For example, the word 'VERB' is *mentioned* when entered into the category L-CAT of lexical category names and is subsequently *used* to refer to the category of verbs (see Sect. 2.5.2). The word 'GOOSE' is *mentioned* in the example sentence of Section 2.6 when specifying that its number is singular, but, in a similar sentence, the word 'GRADY' is *used*.

As a more sophisticated example combining use and mention, we illustrate our system's processing of an equivalent version of the classic sentence of Tarski (1944) "'SNOW IS WHITE' IS TRUE IF AND ONLY IF SNOW IS WHITE". We do not treat truth relative to possible worlds. Our semantic network represents only the belief space of the system, and asserted propositions are those believed by the system.

We continue to build upon the language definition thus far input to the system in this article. The additional lexical entries that we input are:

L-CAT → 'MASS-NOUN ADJECTIVE → 'TRUE
PROPERTY → 'TRUTH-VALUE ADJECTIVE → 'FALSE
MASS-NOUN → 'SNOW

We explain to the system that

TRUE IS A TRUTH-VALUE
FALSE IS A TRUTH-VALUE

to be parsed and interpreted by the system as PROPERTY-CLASS-ENTRIES as shown in Section 2.6. Additional syntax rules such as the following are needed:

NOUN-PHRASE → MASS-NOUN
NOUN-PHRASE → LITERAL-STRING

Upon input of the sentence

IF SNOW IS WHITE THEN "SNOW IS WHITE" IS TRUE

the system builds the parse tree shown in Figure 30 for the utterance.

Applying the teacher's rules, the system builds the network rule of Figure 31 as the interpretation of the input sentence. Node M92 represents the generic string "SNOW IS WHITE" and not just an instance of the string. Node M92 dominates a pattern that is matched by any instance of the string, with V8 a universally quantified variable node.

If the system believes that snow is white, then the rule shown in Figure 31 is used appropriately and if we query the system regarding any instance of the string "SNOW IS WHITE" it indicates that the string is true.

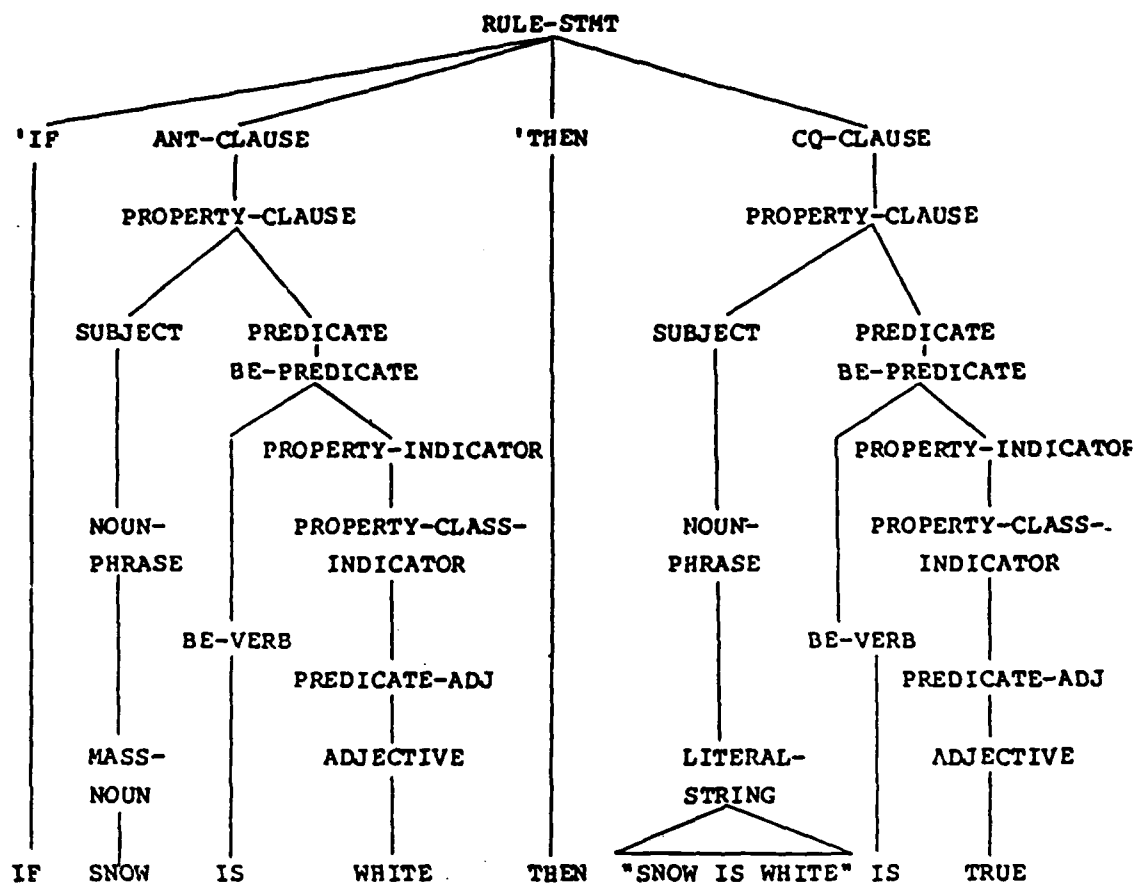


Fig.30. Annotated parse tree

To complete the original bi-conditional statement, the converse statement

IF "SNOW IS WHITE" IS TRUE THEN SNOW IS WHITE

can also be entered the system and the converse of the rule of Figure 31 is built into the network as its interpretation.

5 Summary

This article has presented our approach to NLU: an approach that focuses on the capability of a natural language to be used as its own metalanguage. It is essential to this approach to have the system's parsing and linguistic knowledge be an integral part of its domain knowledge. It is our view that linguistic knowledge about a word or phrase is a part of its meaning or significance and, furthermore, there is no clear boundary line separating syntactic, semantic, and world knowledge. For these reasons we represent linguistic knowledge along with other domain knowledge in an integrated knowledge base. Furthermore, the linguistic rules of the system's knowledge base comprise the system's knowledge of language understanding in the same way that the rules of any rule-based system comprise that system's

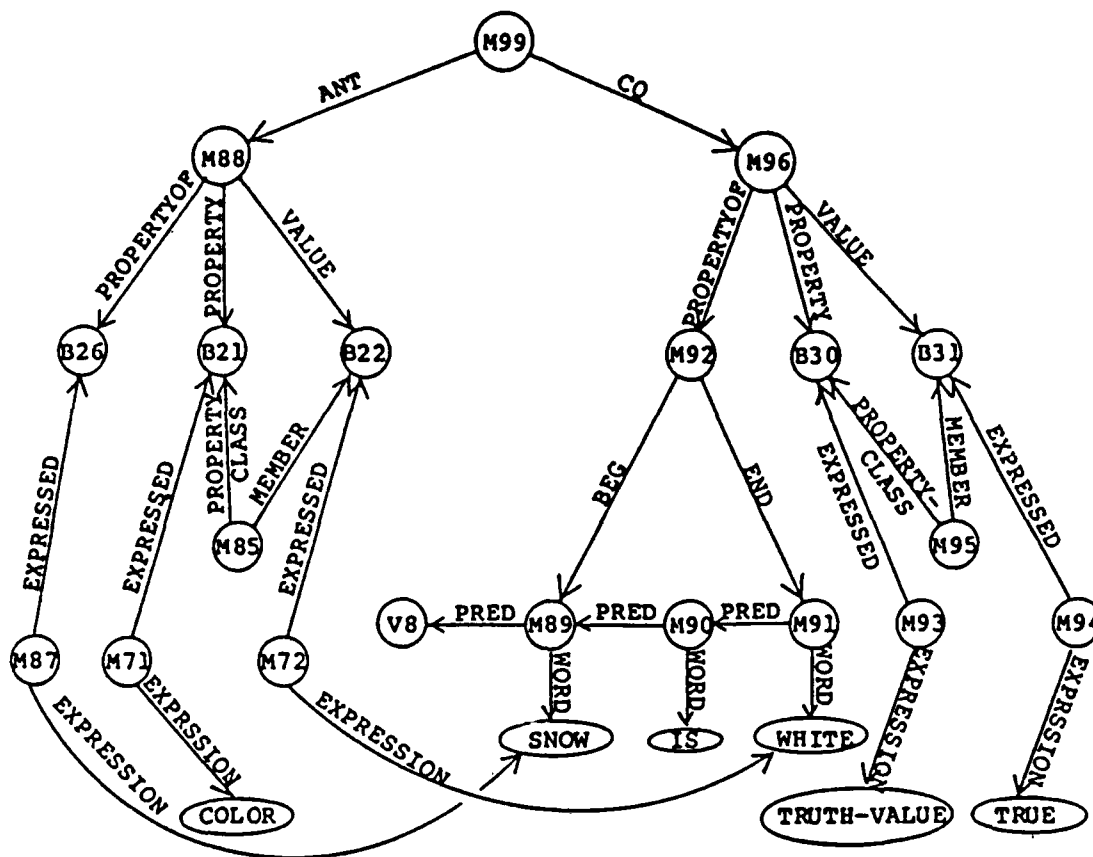


Fig. 31. Interpretation of the input utterance "IF SNOW IS WHITE THEN 'SNOW IS WHITE' IS TRUE"

knowledge of its domain of application. Our system also incorporates the use-mention distinction for language.

We are exploring the possibility of a NLU system's becoming more adept in its use of some language by being instructed in the use of the language. We wish this explanation to be given in an increasingly sophisticated subset of the language being taught. The system must start with some language facility, and we are interested in seeing how small and theory-independent we can make the initial kernel language.

In this chapter, we have discussed the core knowledge and representations of our system, including the kernel language, which consists of predefined terms, and syntactic and semantic rewrite rules with which to bootstrap into a more sophisticated language definition. We have demonstrated the capability of increasing the system's language facility by using the very same facility to instruct the system about language understanding. We built up the system's capability to the stage at which it processed the sentence "IF THE HEAD-NOUN OF A NOUN-PHRASE X HAS NUMBER Y THEN X HAS NUMBER Y". We presented additional examples of language being treated as the topic of discourse including the system's parsing and interpretation of the sentence "'SNOW IS WHITE' IS TRUE IF AND ONLY IF SNOW IS WHITE".

We discussed the system's parsing strategy, which is a combined bottom-up, top-down strategy. Our system's parser is a general rule-based inference system in which applicable rules are activated in parallel in the form of processes or demons. The inference system employs bi-directional inference to cut down the fan out of pure forward or backward chaining.

Acknowledgements. We would like to thank the SNePS Research Group of the State University of New York at Buffalo for their constructive comments and opinions during the course of this research. In particular, we thank William J. Rapaport for his comments on an earlier version of this chapter.

Appendix *Chronological Summary of Input to the System as Presented in This Chapter*

Section Number	Input
2.5.2	L-CAT → 'NOUN
2.5.2	L-CAT → 'PROPER-NOUN
2.5.2	L-CAT → 'DEF-DET
2.5.2	L-CAT → 'INDEF-DET
2.5.2	L-CAT → 'VERB
2.5.2	L-CAT → 'BE-VERB
2.5.2	L-CAT → 'ADVERB
2.5.2	L-CAT → 'ADJECTIVE
2.5.2	L-CAT → 'PREPOSITION
2.5.2	L-CAT → 'CONJ
2.5.2	L-CAT → 'PROPERTY
2.5.2	S-CAT → 'HEAD-NOUN
2.5.2	S-CAT → 'STRING
2.5.2	VARIABLE → 'X
2.5.2	VARIABLE → 'Y
2.5.2	NOUN → 'GOOSE
2.5.2	NOUN → 'GEESE
2.5.2	PROPER-NOUN → 'GRADY
2.5.2	PROPER-NOUN → 'GLADYS
2.5.2	DEF-DET → 'THE
2.5.2	INDEF-DET → 'A
2.5.2	VERB → 'HAS
2.5.2	BE-VERB → 'IS
2.5.2	ADVERB → 'THEN
2.5.2	ADJECTIVE → 'WHITE
2.5.2	ADJECTIVE → 'SINGULAR
2.5.2	ADJECTIVE → 'PLURAL
2.5.2	PREPOSITION → 'OF
2.5.2	CONJ → 'IF
2.5.2	PROPERTY → 'COLOR
2.5.2	PROPERTY → 'NUMBER
2.5.2	UNIQUE-MEANING-CAT → 'ADJECTIVE
2.5.2	UNIQUE-MEANING-CAT → 'PROPERTY
2.5.2	PROPERTY-CLAUSE → SUBJECT PREDICATE

Section Number	Input
2.5.2	SUBJECT → NOUN-PHRASE
2.5.2	NOUN-PHRASE → LITERAL
2.5.2	NOUN-PHRASE → VARIABLE
2.5.2	NOUN-PHRASE → PROPER-NOUN
2.5.2	PREDICATE → RELATION-PREDICATE
2.5.2	PREDICATE → BE-PREDICATE
2.5.2	RELATION-PREDICATE → RELATION PREDICATE-ADJ
2.5.2	BE-PREDICATE → BE-VERB PROPERTY-INDICATOR
2.5.2	RELATION → 'HAS PROPERTY-INDICATOR
2.5.2	PROPERTY-INDICATOR → PROPERTY-CLASS-INDICATOR
2.5.2	PROPERTY-INDICATOR → PROPERTY
2.5.2	PROPERTY-CLASS-INDICATOR → PREDICATE-ADJ
2.5.2	RULE-STMT → 'IF ANT-CLAUSE THEN CQ-CLAUSE
2.5.2	RELATION PREDICATE-ADJ → RELATION ADJECTIVE
2.5.2	RELATION PREDICATE-ADJ → RELATION VARIABLE
2.5.2	BE-VERB PREDICATE-ADJ → BE-VERB ADJECTIVE
2.5.2	'IF ANT-CLAUSE → 'IF PROPERTY-CLAUSE
2.5.2	THEN CQ-CLAUSE → THEN PROPERTY-CLAUSE
2.5.3	PROPERTY-CLAUSE :: PROPERTY SUBJECT PROPERTY PROPERTY-INDICATOR VALUE PREDICATE-ADJ
2.5.3	ANT-CLAUSE :: PROPERTY-CLAUSE
2.5.3	CQ-CLAUSE :: PROPERTY-CLAUSE
2.5.3	SUBJECT :: NOUN-PHRASE
2.5.3	NOUN-PHRASE :: LITERAL
2.5.3	NOUN-PHRASE :: VARIABLE
2.5.3	NOUN-PHRASE :: PROPER-NOUN
2.5.3	PROPERTY-INDICATOR :: PROPERTY-CLASS-INDICATOR
2.5.3	PROPERTY-INDICATOR :: PROPERTY
2.5.3	PREDICATE-ADJ : ADJECTIVE
2.5.3	PREDICATE-ADJ :: VARIABLE
2.5.3	PROPERTY-CLASS-INDICATOR > > [MEMBER ADJECTIVE PROPERTY-CLASS +]
2.5.3	INDEF-S-PHRASE > > [BSTR STRING CAT S-CAT STRC +]
2.6	GRADY HAS COLOR WHITE
2.6	'GOOSE HAS NUMBER SINGULAR
2.6	PROPERTY-CLASS-ENTRY → ADJECTIVE 'IS 'A PROPERTY
2.6	PROPERTY-CLASS-ENTRY :: MEMBER ADJECTIVE PROPERTY-CLASS PROPERTY
2.6	WHITE IS A COLOR
2.6	PLURAL IS A NUMBER
2.6	GLADYS IS WHITE
2.6	'GEESE IS PLURAL
3.2	DEF-S-PHRASE → DEF-DET S-CAT
3.2	INDEF-S-PHRASE → INDEF-DET S-CAT
3.2	MAIN-APPOS-PHR VAR-NAME → INDEF-S-PHRASE VARIABLE
3.2	VAR-APPOSITION-PHR → MAIN-APPOS-PHR VAR-NAME
3.2	SUP-STRING-REF → VAR-APPOSITION-PHR
3.2	CONSTIT-REF → DEF-S-PHRASE
3.2	CONSTIT-PHRASE → CONSTIT-REF 'OF SUP-STRING-REF
3.2	NOUN-PHRASE → CONSTIT-PHRASE

Section Number	Input
3.2	CONSTIT-PHRASE > > [CONSTIT + CONSTITOF SUP-STRING-REF] [BSTR STRING CAT DEF-S-PHRASE STRC +]
3.2	SUP-STRING-REF :: VAR-APPOSITION-PHR
3.2	MAIN-APPOS-PHR :: INDEF-S-PHRASE
3.2	NOUN-PHRASE :: CONSTIT-PHRASE
3.2	DEF-S-PHRASE :: S-CAT
3.3	IF THE HEAD-NOUN OF A NOUN-PHRASE X HAS NUMBER Y THEN X HAS NUMBER Y
4	L-CAT → 'MASS-NOUN
4	PROPERTY → 'TRUTH-VALUE
4	MASS-NOUN → 'SNOW
4	ADJECTIVE → 'TRUE
4	ADJECTIVE → 'FALSE
4	TRUE IS A TRUTH-VALUE
4	FALSE IS A TRUTH-VALUE
4	NOUN-PHRASE → MASS-NOUN
4	NOUN-PHRASE → LITERAL-STRING
4	IF SNOW IS WHITE THEN "SNOW IS WHITE" IS TRUE
4	IF "SNOW IS WHITE" IS TRUE THEN SNOW IS WHITE

References

- Allen, J. (1978): *Anatomy of LISP*. McGraw-Hill, New York
- Bever, T.G. (1970): The Cognitive Basis for Linguistic Structures. In: Hayes, J. R. (ed.): *Cognition and the Development of Language*. Wiley, New York pp.279-352
- Bever, T.G. (1973): Serial Position and Response Biases do Do Not Account for the Effect of Syntactic Structure on the Location of Brief Noises During Sentences. *Journal of Psycholinguistic Research* 2, 187-288 (1973)
- Bobrow, R.J. (1978): The RUS System. BBN Report No.3878
- Bobrow, R.J. and Webber, B. (1980): Knowledge Representation for Syntactic/Semantic Processing. *Proc. AAAI-80*, pp.316-323
- Brachman, R.J. (1977): What's in a Concept: Structural Foundations for Semantic Networks. *Int. J. Man-Machine Studies* 9, 127-152 (1977)
- Brachman, R.J. (1978a): A Structural Paradigm for Representing Knowledge. BBN Report No.3605
- Brachman, R.J., Ciccarelli, E., Greenfeld, N., and Yonke, M. (1978b): KLONE Reference Manual. BBN Report No.3848, July 1978
- Brachman, R.J. (1979): On the Epistemological Status of Semantic Networks. In Findler, N. (ed.): *Associative Networks*. Academic Press, New York, pp.3-50
- Brown, G. and Yule, G. (1983): *Discourse Analysis*. Cambridge University Press, Cambridge
- Bruce, B. (1975): Case Systems for Natural Language. *Artificial Intelligence* 6, 327-360 (1975)
- Burge, W.H. (1975): *Recursive Programming Techniques*. Addison-Wesley, Reading
- Charniak, E. (1981): A Common Representation for Problem-Solving and Language-Comprehension Information. *Artificial Intelligence* 16 (3), 225-255 (1981)
- Dahl, V. (1979): Quantification in a Three-Valued Logic for Natural Language Question-Answering Systems. *Proc. IJCAI 79*, pp.182-187
- Dahl, V. (1981): Translating Spanish into Logic Through Logic. *AJCL* 7 (3), 149-164 (1981)
- Fillmore, C. (1968) : The Case for Case. In: Bach, E. and Harms, R. (eds.): *Universals in Linguistic Theory*. Holt, Rinehart, and Winston, pp.1-90

- Fodor, J.A. and Garrett, M.F. (1967): Some Syntactic Determinants of Sentential Complexity, *Perception and Psychophysics* 2, 289-296 (1969)
- Hayes, P. (1977): On Semantic Nets, Frames and Associations. *Proc. IJCAI 77*, pp.99-107
- Hendrix, G.G. (1978): The Representation of Semantic Knowledge. In: Walker, D.E. (ed.): *Understanding Spoken Language*. Elsevier North-Holland, Amsterdam
- Hendrix, G.G. (1979): Encoding Knowledge in Partitioned Networks. In: Findler, N. (ed.): *Associative Networks*. Academic Press, New York
- Kaplan, R.M. (1973): A Multi-processing Approach to Natural Language. *Proceedings of the National Computer Conference*. AFIPS Press, Montvale, NJ, pp.435-440
- Kay, M. (1973): The Mind System. In Rustin, R. (ed.): *Natural Language Processing*. Algorithmics Press, New York, pp.153-188
- Levelt, W.J.M. (1970): Hierarchical Chunking in Sentence Processing. *Perception and Psychophysics* 8, 99-102 (1970)
- Levelt, W.J.M. (1974): *Formal Grammars in Linguistics and Psycholinguistics*, Vol.3: Psycholinguistic Applications. Mouton, The Hague
- Maida, A.S. and Shapiro, S.C. (1982): Intensional Concepts in Propositional Semantic Networks. *Cognitive Science* 6, 4 (1982)
- McCord, M.C. (1982): Using Slots and Modifiers in Logic Grammars for Natural Language. *Artificial Intelligence* 18 (3), 327-367 (1982)
- McKay, D.P. and Shapiro, S.C. (1980): MULITI - A LISP Based Multiprocessing System. *Conference Record of the 1980 LISP Conference*, Stanford University, pp.29-37
- Pereira, F.C.N. and Warren, D.H.D. (1980): Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13, 231-278 (1980)
- Pollack, J., and Waltz, D. (1982): Natural Language Processing Using Spreading Activation and Lateral Inhibition. *Proc. Conf. of Cognitive Science Society*, pp.50-53
- Quillian, R. (1968): Semantic Memory. In: Minsky, M. (ed.): *Semantic Information Processing*. MIT Press, Cambridge
- Quillian, R. (1969): The Teachable Language Comprehender: A Simulation Program and the Theory of Language. *CACM* 12, 459-476 (1969)
- Quine, W.V. (1951): *Mathematical Logic*. Harper and Row
- Quine, W.V. (1948): On What There Is. *Review of Metaphysics* 2. Reprinted in: Linsky, L. (ed.): *Semantics and the Philosophy of Language*. University of Illinois Press, Chicago, 1952, pp.189-206
- Robinson, J.A. (1965): A Machine-oriented Logic Based on the Resolution Principle. *JACM* 12, 23-41 (1965)
- Roussel, P. (1965): Prolog: Manuel de Reference et d'Utilisation. Groupe d'Intelligence Artificielle, Universite de Marseille-Luminy, September, 1975
- Rumelhart, D. and Norman, D. (1973): Active Semantic Networks as a Model of Human Memory. *Proc. IJCAI 73*, pp.450-457
- Schubert, L. (1976): Extending the Expressive Power of Semantic Networks. *Artificial Intelligence* 7 (2), 163-198 (1976)
- Schubert, L.K., Goebel, R.G., and Cercone, N.J. (1979): The Structure and Organization of a Semantic Net for Comprehension and Inference. In: Findler, N. (ed.): *Associative Networks*. Academic Press, New York
- Schubert, L.K. and Pelletier, F.J. (1982): From English to Logic: Context-Free Computation of 'Conventional' Logical Translation. *AJCL* 8 (1), 26-44 (1982)
- Shapiro, S.C. (1971): A Net Structure for Semantic Information Storage, Deduction and Retrieval. *Proc. IJCAI 71*, pp.512-523
- Shapiro, S.C. (1979a): The SNePS Semantic Network Processing System. In: Findler, N. (ed.): *Associative Networks - The Representation and Use of Knowledge by Computers*. Academic Press, New York, pp.179a-203
- Shapiro, S.C. (1979b): Using Non-Standard Connectives and Quantifiers for Representing Deduction Rules in a Semantic Network. Invited paper presented at Current Aspects of AI Research, a seminar held at the Electrotechnical Laboratory, Tokyo
- Shapiro, S.C. and the SNePS Implementation Group (1981): *SNePS User's Manual*. Department of Computer Science, SUNY at Buffalo, NY

- Shapiro, S.C. and Neal, J.G. (1982): A Knowledge Engineering Approach to Natural Language Understanding. Proc. ACL, pp.136-144
- Shapiro, S.C., Martins, J., and McKay, D. (1982): Bi-Directional Inference. Proc. of the Cognitive Science Society, pp.90-93
- Simmons, R. (1973): Semantic Networks: Their Computation and Use for Understanding English Sentences. In: Schank, R. and Colby, K. (eds.): Computer Models of Thought and Language. Freeman
- Tarski, A. (1944): The Semantic Conception of Truth. Philosophy and Phenomenological Research 4. Reprinted in: Linsky, L. (ed.): Semantics and the Philosophy of Language. University of Illinois Press, Chicago, 1952, pp.13-47
- Warren, D.H.D., and Pereira, F.C.N. (1982) An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *AJCL* 8 (3-4), 110-119 (1982)
- Woods, W.A. (1975): What's in a Link: Foundations for Semantic Networks. In: Bobrow, D.G. and Collins, A.M. (eds.): Representation and Understanding. Academic Press, New York, pp.35-82

Appendix B3:

“Design of an Incremental Compiler for a Production-system ATN Machine”

Yuhan, A.H.
Shapiro, S.C.

**DESIGN OF AN INCREMENTAL COMPILER FOR A
PRODUCTION-SYSTEM ATN MACHINE**

A. Hanyong Yuhan, Stuart C. Shapiro

87-09

July 1987

**Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, New York 14260**

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

DESIGN OF AN INCREMENTAL COMPILER FOR A PRODUCTION-SYSTEM ATN MACHINE¹

A. Hanyong Yuhan
and
Stuart C. Shapiro

Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, New York 14260

ABSTRACT

We present a new design technique for constructing an incrementally-built, compiled, production system ATN machine. This technique, the *re-entrant state module technique*, makes it possible to have a highly modularized, compiled ATN machine that allows flexible search control by having a variable entry point included in the configuration frame of every state module. We have shown this solution to be practically feasible by implementing an efficient ATN compiler of this design.

1. INTRODUCTION

The properties of the *Augmented Transition Network* (ATN) formalism have been intensively investigated by a number of researchers [Thorne, Bratley & Dewar (1968), Bobrow & Fraser (1969), Woods (1970; 1973; 1978; 1980), Kaplan (1972; 1973a; 1973b; 1975), Bates (1978)]. The expressive power of the ATN formalism is at least as great as that of transformational grammars. The recursive nature of an ATN's use of PUSH arcs to find non-terminal constituents gives it the power of handling phrase-structure rules, while its register handling capability, including the effects exhibited by VIR arcs coupled with the HOLD action, gives it the power of handling context-sensitive transformational rules. For this reason, ATNs have been widely used to express parsing grammars for many formal or natural language processing

¹ This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under contract No. F30602-85-C-0008. Bill Rapaport's valuable comments on earlier drafts of this paper were greatly appreciated.

systems [Woods, Kaplan & Nash-Webber (1972), Woods et al. (1976), Waltz (1976), Bobrow et al. (1977), Wanner & Maratsos (1978), Kwasny & Sondheimer (1981), McKay & Martins (1981), Weischedel & Sondheimer (1984), Shapiro (1982), Shapiro & Rapaport (1985)]. Shapiro (1975; 1982) has shown that a single ATN formalism can be used to express grammars for both sentence recognition and sentence generation; thus, ATNs may be used for bidirectional symbol transductions between surface linguistic forms and internal semantic representations.

Since hardware ATN machines are not yet available, ATNs are processed by ATN virtual machines emulated by software. Usually, ATN programs (grammars) are processed by an ATN interpreter, another program usually written in Lisp. An ATN interpreter is a general-purpose ATN virtual machine in the sense that it can be used to run any ATN program. However, systems using ATN interpreters tend to be very slow. Another way of building an ATN virtual machine is, to combine a given ATN program with the emulating program into one homogeneous unit. However, such an ATN machine is only good for the ATN program it was made for. We call an ATN machine of this kind a *compiled ATN machine*. By an *ATN compiler*, we mean a special device (or program) that builds a compiled ATN machine when a particular ATN program is given. The input/output relations for the various (virtual) machines we have defined are depicted in Figure 1.

Figure 1. Input/output relations of various
ATN-related machines

The obvious advantage of compiling an ATN is faster execution, although this is probably at the expense of a larger program space.² However, in most computing environments, space problems are usually efficiently buffered by the operating system, leaving users with no significant penalty for this trade-off. Therefore, designing a good ATN compiler is an interesting issue to be addressed in computational linguistics. In this paper, we first examine the properties of an ATN parser as an abstract machine and compare two different models for ATN machines. We

² Finin (1983:37) reports that he obtained a size reduction of the program space as well through his ATN compilation.

then analyze the advantages and problems associated with various approaches to ATN compilers and present a solution to a problem faced by what is arguably the best approach. Finally, we provide a brief description of an ATN interpreter due to Kwasny and Shapiro (henceforth, the Kwasny-Shapiro interpreter that we believe is a suitable ATN virtual machine by the standards discussed here, and present our implementation of an ATN compiler based on that interpreter. However, the applicability of our design technique for the compiler is not limited to the Kwasny-Shapiro interpreter.

2. TWO MODELS OF ATN MACHINES

When an ATN program G for a language L is appropriately loaded, an ATN machine MG tests whether an input S is a sentence of L , possibly producing an analysis of S as a side-effect if S is a sentence of L . At this point, it is not our immediate concern whether MG is an actual machine or a virtual machine implemented by a program running on another machine. Neither does it concern us whether G is explicitly loaded into MG or is a part of MG itself. We view an *ATN program* as an expression of a search space consisting exclusively of OR-trees. Each ATN state is a node, with the initial state being the root node. Each arc from a given ATN state is an edge from the node for the state.³ If S is a sentence of L , it is mapped to a set of paths from the root to a terminal in the search space, where each such path represents a unique analysis of S . If S is not a sentence of L , it fails to be mapped to any path starting from the root to a terminal. On this view, a process of the ATN machine's computation, which we will call *ATN parsing*, is essentially an OR-tree search for a solution path or a set of solution paths in the search space defined by G , where the search is

³ One may alternatively view the search space made by an ATN program as a directed graph in which a single node can be reached through more than one path. The parsing technique of keeping a substitution table for well-formed substrings appears to be based on this view [Tennant (1981: 67, 270)]. However, whether a string analyzed into a certain syntactic structure will have an invariant semantic interpretation regardless of its context is as yet an open question. If the purpose of parsing is more than a mere acceptability test of the input form, it may sometimes turn out to be inadequate to associate a well-formed substring with the results recorded in the table due to a previously parsed substring which was found in a different path environment. We will not pursue this issue here, since it is not directly relevant and our model is neither dependent on nor incompatible with this view.

conditioned by the input S .

The central component of an ATN machine is the part that controls the search in this OR-tree search space. There are two different models for managing this control on a sequential machine: the *host-recursion model* and the *production-system model*.

The host-recursion model utilizes the host machine's recursive control and internal stack. Here, each search step in the ATN search space, such as a transition from a state $S1$ to another state $S2$ via an arc C , is realized as a direct call to the program module corresponding to C from the program context for $S1$ in order to create a program context for $S2$ embedded one level further down in the control stack. Since an ATN state transition is realized as a recursive call at the host machine level, the configuration frame⁴ of each ATN state can be maintained in the host system's run-time stack of activation records. For this reason, the ATN machine's backtracking is transparently managed by the host machine's own recursive control and its run-time stack. A depth-first search strategy can easily be implemented in a host-recursion ATN machine. This method makes the data flow and control flow of an ATN machine very simple and effortless. But its drawback, besides the potential danger of the host machine's stack overflow, is that the control of the ATN search is tied to the host programming language's data and control structures. Because of this, it becomes relatively more difficult to allow the ATN user to exercise control over the order of search during parse time. The implementation of multiple parsing on a sequential machine also turns out to be quite cumbersome in this model because of the unnatural forcing of backtracking after a successful return of a subroutine call.⁵

In a production-system model, on the other hand, the control driver of an ATN machine is an indefinitely iterated execution of search processes. It stops only when an interrupt flag is set because either the search processes have been exhausted or the goal of the computation has

⁴ By this, we mean the ATN-level data objects relevant to a state such as the state name, the present input buffer, the path history, etc., contrasted to the data objects meaningful only at the ATN machine emulation level.

⁵ This is usually achieved by the host language's facility for a forced backtracking, such as THROW and CATCH found in some Lisps [Christaller (1983), Laubach & Barth (1983)].

been achieved. This is an adaptation of the standard production system architecture [Post (1943), Davis & King (1977: 30)]. In each cycle of the production, a process is taken out of the process queue and executed. The effect of a process being executed is the realization of a further step of the search in the OR-tree search space, possibly resulting in the creation of a number of new successor processes appropriately inserted in the process queue according to their respective search priority. Each process has a configuration frame that specifies the actions to be taken for the particular search step and the data on which those actions are to be applied. Every arc of each ATN state is realized as such a process, and an ATN state is realized as a group of processes that share the same configuration frame except for the actions to be taken. An ATN machine of this model keeps possession of the top-level control of the ATN parsing because of explicit maintenance of the process queue and the configuration frames for the processes in the queue. Since the process queue is maintained at parse time, this model allows the order of the search to be controlled flexibly during parse time by a priority policy for process scheduling. Thus, the order of the search is, in principle, dynamically determined at the time when an ATN program runs, rather than at the time when the program is loaded into the ATN machine.

According to Winograd (1983: 261), "*the scheduling strategy* is an important theoretical property of the ATN formalism. Unlike the parsers that are tied to a specific processing order ..., an ATN parser can make use of a grammar that was written without a specific processing regime in mind." Thus, an ATN machine with no capability for managing its own scheduling control can probably be considered as theoretically incomplete, despite Finin's (1983) defense of his depth-first backtracking implementation of an ATN machine. Therefore, even if a particular implementation does not elaborate a policy of scheduling priority for ATN parsing and ends up with an extremely trivial algorithm that makes it, in effect, nothing more than a depth-first search, we believe that the basic architecture of an ATN machine should be designed in such a way that it is capable of incorporating flexible search control when desired. For this, we conclude that the choice between these two ATN models is more a practical issue than a

theoretical one, and we believe that choosing the architecture of an ATN machine with a pre-determined search control by choosing the host-recursion model is comparable to choosing a "quick and dirty" solution.⁶

The advantage of the production-system model over the host-recursion model does not stop at the ATN machine's capability for dynamic scheduling of the search order at parse time. Since each search process is represented in the machine declaratively rather than procedurally, it is also very simple in this model to incorporate a debugging facility to be used to monitor the behavior of the parsing, as will be shown in our later discussion.

3. EVOLUTION OF ATN COMPILER CONSTRUCTION

The approach to ATN compiler construction has been changing from the earlier *integral compilation* approach to the present *incremental compilation* approach. In the integral compilation approach [Burton (1976), Burton & Woods (1976)], the compiler compiles an ATN program into one huge fragment of an ATN object program, which is characteristically a single loop around a case statement (or a computed GOTO-statement), where, in essence, each ATN state is a label for a segment of code that implements all the arcs emanating from that state.⁷ On the other hand, the incremental compilation approach [Finin (1977; 1983), Christaller (1983), Kochut (1983), Laubsch & Barth (1983)] attempts a maximum modularization of the ATN object program in such a way that each ATN module (such as an ATN state or arc) is mapped to an associated program module in the compiled code. The relation between these two approaches parallels the one between the spaghetti-GOTO style of programming and the structured programming.

The obvious advantages of incremental compilation over integral compilation are the clarity of the compiled code in the host language and the greater maintainability of the compiled ATN program. Furthermore, because the code segments implementing each ATN state or

⁷ For a good sketch of this approach, see Laubsch & Barth (1983: 152-162).

are detached from the code segment for the control driver, the perspicuity of the ATN notation is better preserved in the compiled ATN program. Since each ATN module can be compiled and loaded independently, the ATN programmer can simply modify a selected part of his source ATN program, and recompile and load those particular modules, leaving the rest intact. Because of this flexibility, even compilation of an extremely large ATN program can be done piece by piece on a small system where the total available space may be restricted. Furthermore, debugging flags or a monitor can very easily be inserted in appropriate object modules corresponding to specific source modules of the ATN, enabling the user to monitor or modify the ATN program in debugging mode while running compiled code.

In general, however, excessive modularization may often sacrifice some of the program's execution efficiency. Depending on the host language, an invocation of a module may sometimes waste its resources creating an activation record for the sake of a formal syntactic requirement. This might have been unnecessary if it were a mere jump within one module. A segment of a program is justified to become a module if a significant amount of local data preparation has to be done on entering its execution context due to the logic of the task. In this light, we argue that modularization of an object ATN program for ATN source modules is not a case of excessive modularization. Every program component corresponding to an ATN source module is expected to have to set up its own data environment anyway. In fact, every compiled ATN program built in the integral approach has done this. Furthermore, if the range of control transfer is widely spread over a number of mutually exclusive alternatives in a large program space, and the operating system on the host machine performs virtual memory management for program space, the efficiency of main memory management can be greatly enhanced by a well-modularized program. This is because the logical segmentation of the program space can help the swapping manager deal with naturally partitioned memory blocks. In fact, because function calls are such a natural form of program control in some stack-oriented languages like Lisp, the cost of module invocation should not deter us from creating logically well-motivated modules.

Considering these factors, we regard the development from integral compilation to incremental compilation as a natural consequence of the evolution of general programming style. The rest of our discussion will be limited to issues and problems encountered in the incremental compilation approach.

4. CURRENT ISSUES IN THE INCREMENTAL APPROACH

A number of designs for incremental ATN compilers have been presented [Finin (1983), Christaller (1983), Laubsch & Barth (1983), Kochut (1983)]. All of them clearly address the problems of integral compilation and succeed in constructing incremental compilers. However, they all share the same pitfall of being too closely tied to the peculiarity of the host language, namely, Lisp. These compiler designs are all host-recursion models. Each ATN state is mapped to a program module in the compiled code, but each transition from one ATN state to another, that is, each step of the ATN search, is realized as a call of one Lisp function from another Lisp function. The basic schema of a compiled ATN state module is an OR or AND construction⁸ over a number of code segments intended for each of the alternative arcs. The consequent limitations are, of course, all those exhibited by ATN machines built on the host-recursion model. One typical shortcoming of this relinquishing of the ATN's control to the host language is exhibited by Laubsch & Barth (1983). Their compiled ATN machine does not handle HOLD actions and alternative lexical interpretations, and does not easily provide users with flexible debugging information. Although Finin (1983:9) argues that he had purposely "chosen to implement a depth-first backtracking version", we feel that there was hardly any other choice left once the ATN machine had surrendered its control to the host language (Lisp, in his case). Due to the direct use of Lisp's own run-time stack and controls (such as OR, AND, THROW, and CATCH) to guide the ATN search, users were deprived of the freedom of toggling the ATN machine between single parsing and multiple parsing modes. Once the single

⁸ OR or AND is selected depending on whether one or all possible parsings of the input sentence are sought. In an actual implementation, AND construction can be replaced by a PROG structure.

parsing policy is chosen as the primary option, allowing multiple parsing turned out to be a problem because a faked failure has to be created even after a genuine success of a series of embedded function calls corresponding to an acceptable ATN search.

Although host-recursion implementations take an incremental compilation approach, in principle, there has been little interest shown in breaking down a state module further into submodules corresponding to each arc of the state. We will call this *arc-module modularization*. We realize that such implementations can achieve a superficial arc-module modularization merely by making a slight modification to the compiler so that it would decompose the code for every ATN state into separate submodules for its arcs. However, we suspect that they are right not to do so in the absence of a compelling motivation, since it would mean even faster consumption of Lisp's run-time stack. In their design, control over the alternative arcs of an ATN state is determined by the compiler once and for all, and then is frozen inflexibly in the program body of the emulating host system. When no control flexibility of invocation is preserved, *arc-module modularization for the sake of mere code decomposition* turns out to be a question about programming style rather than an issue of theoretical significance.

Earlier, we pointed out that the production-system model is a better solution for an ATN machine that preserves its own search control. On this design, the ATN machine explicitly maintains its own appropriately sorted process queue. To our knowledge, there have been no discussions of the design issues of compiled ATN machines of the production-system model viewed from the incremental compilation approach. At first thought, it does not appear to be too difficult to construct a production-system-like driver that would work on a queue of processes consisting of ATN state modules that would be pretty much like the state modules in the host-recursion model. But if we decide to map an ATN state, rather than an arc, to the smallest ATN module in the compiled ATN machine, we are faced with a serious problem. Granting that a compiled program cannot be modified during run-time, it looks as if we have no other choice but to freeze the ATN's search order within a state at the time of ATN program compilation. Since every arc emanating from each ATN state is mapped to an ATN

later re-entrance for the remaining alternative arcs. The problem is reduced to that of how a state module can be made re-entrant.

5. THE RE-ENTRANT STATE MODULE TECHNIQUE

In the Kwasny-Shapiro interpreter based on the production system model, each process in the queue is made of a configuration frame that holds the source definition of an ATN state⁹ being active in the search frontier, as well as other information relevant to a process. Every time a process is fired, one and only one of its arc alternatives is consumed, and the alternative arc list of the process is appropriately reduced. When a process is on the queue, it corresponds to a state module representing all the alternative arcs. However, when it is fired, a process is realized as an arc module that activates a specific search process associated with a particular arc instance. We call processes on the queue, each mapped to an ATN state active on the search frontier, *archiprocesses*¹⁰ to distinguish them from actually expanded, specific individual search processes. When an archiprocess is fired, it chooses one of its successor processors in accordance with a priority policy to create a successor archiprocess, which is then also inserted into the queue. The successor archiprocess inherits its configuration frame, appropriately modified by its own search actions, from the parent process. When such a child process is created, the parent archiprocess also modifies its own configuration frame so that it no longer contains that consumed process in its alternative list, or completely removes itself from the queue if no alternatives are left.

The solution to the problem of devising a re-entrant state module lies in the implicit notion of archiprocesses found in the Kwasny-Shapiro interpreter. The process queue of the compiled ATN machine can also queue a sequence of archiprocesses, but with the configuration frames slightly restructured. In a compiled ATN machine, the configuration frame of an

⁹ A source definition of a state is a set of source definitions of alternative arc instances.

¹⁰ This term is borrowed from, and is analogous to, the notion of "archiphonemes" or "archi-unit" used in phonology, as explained in Chomsky & Halle (1968:64).

search process, a compiled production-system ATN machine is impossible even to imagine without, at least, an implicit notion of arc-module modularization. Without resolving this arc-module modularization problem, it is impossible to make the system have the capability of incorporating flexible search control. One might propose splitting state modules such as the above into stand-alone arc modules for every arc in the state, and queue them in the process queue as if they were totally unrelated to their sibling processes. However, all the sibling arcs from one ATN state are, in fact, closely related to one another by the facts that they inherit a common configuration frame and that they are competing alternatives at a same point of the search context. Because of this, the compiled production-system ATN machine should preserve the ATN states' statehood property, as well. We therefore reject the approach that would eliminate state modules in the compiled machine.

We conclude that, in order to construct a compiled ATN machine which will meet the requirements of capturing the ATN states' statehood and preserve control flexibility over alternative arcs, we will need to map both ATN states and arcs to modules of the compiled production-system ATN machine. The consequent problem is, then, how we can arrange it so that each process in the queue will corresponding simultaneously with a state itself and with only one of its arc instances. This is a problem for the host-recursion model, as well; however, it was not realized to be a problem, since the host system's control automatically stores a state module on the run-time stack and invokes each arc alternative one by one using the host system's backtracking mechanism. This is done at the expense of the ATN's being limited to the blind depth-first search policy (cf. Section 7.3.1). We find the notion of program re-entrance hidden in this mechanism of host system's stack manipulation and automatic backtracking control. A state module remains in the stack waiting for re-entry to each of its arc instance. In the production-system model, an ATN's own process queue is used instead of the host system's stack. Thus, the solution to this problem for a production-system ATN machine seems to be to make each process in the queue be a state module, where a state module is not only consumed and exited from with one arc tried, but also left in the queue available for a

later re-entrance for the remaining alternative arcs. The problem is reduced to that of how a state module can be made re-entrant.

5. THE RE-ENTRANT STATE MODULE TECHNIQUE

In the Kwasny-Shapiro interpreter based on the production system model, each process in the queue is made of a configuration frame that holds the source definition of an ATN state⁹ being active in the search frontier, as well as other information relevant to a process. Every time a process is fired, one and only one of its arc alternatives is consumed, and the alternative arc list of the process is appropriately reduced. When a process is on the queue, it corresponds to a state module representing all the alternative arcs. However, when it is fired, a process is realized as an arc module that activates a specific search process associated with a particular arc instance. We call processes on the queue, each mapped to an ATN state active on the search frontier, *archiprocesses*¹⁰ to distinguish them from actually expanded, specific individual search processes. When an archiprocess is fired, it chooses one of its successor processors in accordance with a priority policy to create a successor archiprocess, which is then also inserted into the queue. The successor archiprocess inherits its configuration frame, appropriately modified by its own search actions, from the parent process. When such a child process is created, the parent archiprocess also modifies its own configuration frame so that it no longer contains that consumed process in its alternative list, or completely removes itself from the queue if no alternatives are left.

The solution to the problem of devising a re-entrant state module lies in the implicit notion of archiprocesses found in the Kwasny-Shapiro interpreter. The process queue of the compiled ATN machine can also queue a sequence of archiprocesses, but with the configuration frames slightly restructured. In a compiled ATN machine, the configuration frame of an

⁹ A source definition of a state is a set of source definitions of alternative arc instances.

¹⁰ This term is borrowed from, and is analogous to, the notion of "archiphonemes" or "archi-unit" used in phonology, as explained in Chomsky & Halle (1968:64).

archiprocess in the queue holds a pointer to an appropriate compiled state module, instead of remembering the source definition of the alternative arcs. A state module is an unalterable program module that contains in it as many distinct entry-points as the number of arcs emanating from the state, making it resemble an ignition distributor of an automobile. Each ATN arc from the state is also mapped to a unique arc module dependent on the state in order to achieve a higher degree of modularization along the lines of the incremental compilation approach. Entering a state module with a specific entry-point value "ignite" a specific arc module. As Figure 2 illustrates, each entrance to a state module results in the invocation of one and only one arc module, and the state module, being left in the queue, remains re-entrant.

 Figure 2. Block-diagram of the control flow through
 a re-entrant state module to an arc module
 ***** ABOUT HERE *****

We claim that this new design technique, the *re-entrant state module technique*, can be used to incrementally build a compiled production-system ATN machine that allows flexible search control at parse time. In this design, as in the Kwasny-Shapiro interpreter, every process in the queue is statically an archiprocess associated with an ATN state currently found in the search frontier, and a firing of such a process is dynamically equivalent to an execution of one and only one alternative arc of the associated state. The priority policy used to decide which arc is to be selected (which is comparable to the cylinder firing order in our automobile analogy) only depends on the algorithm¹¹ that dynamically computes the value of the entry-point for each entrance to the state module.

6. DESCRIPTION OF THE KWASNY-SHAPIRO ATN INTERPRETER

In this section, we describe the organization of the Kwasny-Shapiro ATN interpreter¹²,

¹¹ The investigation and elaboration of this algorithm is certainly an issue related to ATN parsing, but is relevant to our discussion of the design issue. However, it is obvious, at least, that the algorithm must not generate one entry value more than once.

¹² This was initially designed by Stan Kwasny and Stuart C. Shapiro at Indiana University in 1974 to the specifications presented by Woods's (1970, 1973) model [Kwasny (1974)] and has been expanded by the SNePS Research Group at SUNY Buffalo, with a major contribution by Gerard Donlon. It presently includes,

an ATN virtual machine currently implemented in Franz LISP and running, normally in the SNePS¹³ environment on VAX 750s and VAX 780s under the Berkeley Unix 4.3 operating system. There are three reasons for describing this interpreter before the ATN compiler and the compiled ATN machine: (i) it was designed according to the production-system model that we have been defending, (ii) the re-entrant state module designing technique was based on it, and (iii) our compiler implementation was directed toward building a compiled ATN machine functionally equivalent to it.

6.1 ATN formalism

The arcs presently supported by the Kwasny-Shapiro ATN machine are: CAT, WRD, TST, TO, JUMP, PUSH, CALL, RCALL, POP, VIR, and GROUP. The syntax and semantics of the ATN formalism used by this interpreter is described in Shapiro (1982), except for the following modifications. JUMP arcs are allowed to succeed with an empty input buffer, because a JUMP arc consumes no input; this relaxation enables an ATN to capture and express some regularities that can best be handled at the end of a sentence. The test ENDOFSENTENCE and the form LEX are defined and given the same semantics as described in Bates (1978). Besides the CALL arc proposed by Shapiro (1982), another new ATN arc, RCALL, has been added; it has the same syntax as a CALL arc (cf. Shapiro, 1982:14), namely:

(RCALL <state> <form> <test> <preaction or action>*
 <register> <action>* <terminal action>)

The semantics of an RCALL arc differs from that of a CALL arc only in that RCALL locally replaces its whole input buffer with <form>, while CALL globally prepends <form> to the current input buffer. The new RCALL arcs were found to be convenient in writing generation grammars in which one often wants to delimit the scope of the local input buffer for a

besides the ATN parser core, an English morphological analyzer and synthesizer written by Darrel Jov at Indiana, and an input interface written by Hanyong Yuhon at SUNY Buffalo that accepts sentences written in normal English orthographic conventions and takes care of punctuation marks and capitalized sentence-initial words.

¹³ SNePS is a semantic network processing system that can be used to represent, store, manipulate knowledge [Shapiro (1979), Shapiro & Rapaport (1985)].

CALL to one and only one specific semantic network structure. GROUP arcs have been implemented so as to have the same meaning as described in Christaller (1983:75) and Winograd (1983:265). A GROUP arc groups a number of arcs into a set of mutually exclusive alternatives such that if the test of any one member of the set succeeds, the rest are not even tried. A GROUP arc is distinguished from other kinds of arcs in that it is a meta-arc [Christaller (1983:129)] intended to specify the control of search on a sequential machine rather than to specify the substance of a search.

6.2 Organization and control structures

The Kwasny-Shapiro ATN interpreter consists of the following Lisp-implemented components: *I/O handler*, *program/data storage*, *control driver*, and *ATN-operation modules*. The four main subcomponents of the I/O handler are the input sentence reader, the ATN program loader, the lexicon data loader, and the message printer. The message printer, called TALK, prints out trace information in varying levels of details selected by the ATN user. The input sentence reader, PARSEERREAD, reads input sentences typed in normal English orthographic convention. The ATN program loader, ATNIN, and the lexicon data loader, LEXIN, load the ATN program and the user's lexicon into appropriate storage areas; they are implemented using the host Lisp's property lists.¹⁴ The ATN operation modules component consists of Lisp functions that implement each of the ATN operations, including all the ATN arcs (such as CAT, WRD, PUSH, etc.), ATN tests and actions (such as NULLR, SETR, HOLD, TO, JUMP, etc.), and all ATN forms (such as *, LEX, GETF, etc.). Each of these Lisp functions tests, modifies, or retrieves appropriate data elements of the configuration frame of the current process or one of the ancestral or descendant processes to satisfy the semantics of the associated ATN operation. The control driver, consisting of the Lisp functions PARSE, PARSE2, and STEPPARSER, is the component that controls the parsing computation in a production-system-like fashion.

¹⁴ ATNIN and LEXIN store the ATN programmer's or the user's definition of each ATN state and lexical entry as the property value of Lisp atom of the entry name under the property indicators, '=arcs and '=dict, respectively. These storage areas, which are (normally) not modified during parse time, function as the read only memory of the ATN machine.

PARSE initializes the ATN machine, reads an input sentence, and then invokes PARSE2¹⁵ to parse the input form. PARSE2 interacts with STEPPARSER to drive the parsing processes. Each process is comparable to an ATN state with all its ATN arcs not yet attempted saved in the list of alternatives, and those ATN arcs already attempted having been removed. As a data type, each process is a configuration frame made of a record of 10 fields, which contains in it all the information necessary for the execution or trace of the process, such as the ATN state name, definitions of the arcs yet to be tried, the table of ATN registers and their values, the input form in the current buffer, the path information of the current search process, and so on. Initially, PARSE2 designates a process made for the starting ATN state as the current process and calls STEPPARSER. Upon being called by PARSE2 for the current process, STEPPARSER iteratively tries the remaining arcs until either it finds the first one that leads to a successful ATN state transition or no alternative arcs are left.¹⁶ Each trial of such an arc alternative is realized as an invocation of an appropriate arc module¹⁷ (such as CAT, PUSH, etc.) with the source ATN arc definition as its argument and the data in the configuration frame as its environment. An arc module returns either a failure report or the successor process¹⁸. The data of the successor process for alternative arcs is newly made with the information copied from the original definition of the target ATN state stored in the read-only ATN program storage area. Eventually, STEPPARSER returns to PARSE2 a queue of processes (possibly empty) that is the result of appending the processes for the remaining alternative arcs to the successor processes returned by the first successful arc module call. If a non-empty successor process queue is returned from STEPPARSER, PARSE2 designates the first process in the queue as the current process and pushes the remaining processes onto the stack of alternative processes. An empty successor process queue returned to PARSE2 from STEPPARSER indicates

¹⁵ PARSE2 can also be directly invoked from SNePS with a default initialization via a SNePS User Language (SNePSUL) command for ATN parser invocation from within SNePS. In this case, SNePS communicates with the ATN parser by means of the SNePS structures left as side-effects of parsing.

¹⁶ The order of making the choice among the alternatives does not have to be tied to the order in which the arcs are listed in the ATN program, although this interpreter adheres to that order for want of another dependable policy.

¹⁷ Note that the interpreter's arc modules are generic (cf. Section 7.3.4).

¹⁸ Since some arcs such as CAT may be succeeded by more than one process, it is actually a set of successor processes.

that the previously designated current process has been completely blocked and that the system should backtrack one step. In this case, PARSE2 undoes, if necessary, all the side-effects created by the blocked process, pops a process from the stack of alternative processes, and designates it as the current process. PARSE2 iterates this cycle, invoking STEPPARSER until either the accumulated results of the parse warrant the termination of the parsing or, upon receiving STEPPARSER's failure report, it finds that the stack of alternative processes is depleted.

6.3 Handling of the ATN's own recursion and side-effects

The above description shows that the Kwasny-Shapiro ATN interpreter was designed according to the production-system model. It maintains a queue of processes and, for each production system cycle, selects and activates a process out of the queue production-system model, not relying on the host system's recursive stack. In this section, we examine how delayed ATN actions or side-effects are handled.

PUSH-family arcs (namely, PUSH, CALL, and RCALL), which push the ATN's environment one level deeper in the ATN's own recursion stack, are hybrid: they contain in their ATN definition two components, which are executed on two non-adjacent occasions. One part is executed when the process for the arc itself is in control; the other part, which we call the list of *push-pop actions*, remains dormant until the process for a matching pop acquires control and awakens it to be executed. When the process for a member of the PUSH-family is executed, it records the push-pop actions, that is, the list of <action>s and <terminal action>, into a special stack reserved in the configuration frame for this particular purpose, and lets it be inherited by all its successor processes. One of the main tasks of POP arc modules is to pop the top of this stack and execute these push-pop actions.

In designing an ATN machine, the designer may sometimes find that some data operations cannot be confined in one process's configuration frame at an acceptable cost. We say that a data manipulation is done as a side-effect if the effect of the data operation is not confined within a process's configuration frame. On a system that adopts the flexible search strategy, determining which side-effects are to be undone and which are to be kept for each process is an

extremely complicated matter. We have noted that the Kwasny-Shapiro ATN interpreter mostly runs in the SNePS environment in which it is allowed to modify SNePS network structures as a side-effect in the middle of the parsing operation. In general, SNePS network structures are global. In this ATN machine, one of the ten cells of the configuration frame, dedicated to remembering all side-effects, records all the SNePS nodes created by the process. When backtracking takes place, PARSE2, by accessing this cell, can thus delete all the SNePS structures created by the failed process.

7. IMPLEMENTATION

In this section, we present a compiler implemented to test the state-module re-entrant technique we have proposed, showing some details of the compiled ATN machine generated by the compiler. This compiler was designed to produce a compiled ATN machine that would be functionally equivalent to the Kwasny-Shapiro interpreter ATN machine, following the same ATN formalism and running in precisely the same software and hardware environments. For this reason, quite a number of program modules are, in fact, shared by both ATN machines. We will first describe the ATN compiler in order to see how it generates a compiled ATN machine for a given ATN program, and then we will examine the details of the compiled ATN machine as an abstract object.

7.1 Description of the compiler

The ATN compiler is a program package written in Franz LISP, running in the same software and hardware environments as the Kwasny-Shapiro ATN interpreter. Given a file containing an ATN program, it generates a Lisp program which is a compiled ATN machine. The initial output of the ATN compilation is a file of Lisp source code which may be further compiled into a lower-level language code using the general Lisp compiler provided in the operating system environment. As a data type, an ATN program is a set of ATN state definitions. To compile an ATN program, the cycle of reading a state definition and then calling COMPSTATE, the specialist for the state compilation, is iterated until the source ATN

program file has been processed. For each ATN state, COMPSTATE generates a segment of Lisp code that defines an associated state module containing in it as many entry points as the number of arcs emanating from the state. Then, for each arc in the state, COMPSTATE calls COMPARC, which in turn calls an appropriate arc compilation specialist. Figure 3 shows the Lisp code for COMPARC.

Figure 3. Lisp definition of COMPARC function
 structures for the compiled ATN machine.
 ***** ABOUT HERE *****

The arc compilation specialists are named COMP<arcname> (such as COMPCAT, COMPPUSH, COMPJUMP, etc.), where <arcname> is the name of the arc it is specialized for. Each one produces the code of each state-dependent arc module. The function COMPGROUP, the specialist for the GROUP arc compilation, generates the code for a pseudo-state (cf. Section 7.3.4) and recursively calls appropriate arc specialists to generate the codes for the embedded arc modules, allowing arbitrarily deeply embedded GROUP arcs.

The task of an arc compilation specialist is to re-write the associated interpreter's generic arc function (such as CAT, PUSH, etc.) into a state-specific arc module completing as much compile time evaluation and code expansion as possible in the context determined by the source state and the arc instance. A typical example of code expansion is unfolding of loops. Iterations in the generic arc function which are designed to interpret arbitrarily many consecutive ATN source actions are opened and unfolded into a finite sequence of the actual actions found in the given arc instance. Whenever the specialist can obtain a compile time evaluation of a form, it takes one of the two actions: (i) if the form is the condition part of a condition-action construction, then, depending on the obtained value, it either directly proceeds to generate the code for the action part, or totally ignores the condition-action construction; (ii) if the form is not a condition part, then it generates the code replacing the form with the bound value with a quote in front so that it won't be evaluated again by the compiled machine later. For those forms that cannot be evaluated at compile time, the specialist generates the code largely by

duplicating the corresponding segment of the code of the interpreter's generic arc module. Sometimes, the arc compilation specialist may attempt a rough evaluation to simplify the compiled code if it should succeed. By rough evaluation, we mean an evaluation that may succeed at compile time in some special case. An example is the examination of the test part of ATN arcs, which often turns out to be a Boolean constant T. The function ROUGHTEST-T is used to avoid generating code for a parse-time call to the function TEST if the test is simply the constant T in the ATN source which even the compiler can tell the value of. However, we have not yet included any extensive code optimization routines or error checking measures.

7.2 The configuration frame of the ATN machine

The data structures of the compiled ATN machine are based on the interpreter machine's data structures. Both machines maintain a queue of processes. As a data type, each process is a configuration frame, which is a record of 10 information fields. In the interpreter, the configuration frame holds the following information:

- <state>, the name of the current ATN state;
- <arcset>, the definitions of the remaining alternative arcs;
- <buffer>, the input form not yet consumed;
- <regset>, the current ATN registers and their values;
- <hstack>, the stack of hold registers and their values;
- <level>, the level of the ATN recursion;
- <pstack>, the stack of the push-pop action lists;
- <path>, the information on the path history;
- <nodeset>, the SNePS nodes created as the process's side-effect;
- <weight>, information for the search priority.

The structure of a configuration frame used in the compiled machine is the same as the interpreter's, except that <arcset> is here replaced by <entry> for the value of the entry point. Another minor difference is that, in the compiled machine, push-pop actions are no longer a list of source ATN actions. When a PUSH-family process is executed, the sequence of the actions in the list is defined into an equivalent lambda function with a unique name assigned, and the name, instead of the action list, is stored in <pstack>.

7.3 Architecture of the compiled ATN machine

The compiled ATN machine is made of two components: the *core* component, and the *ATN module* component. The ATN module component, which originates in the user's ATN program, consists of all the compiled ATN modules, each of which is either a mapping of an ATN state or an arc instance. The core component includes the control driver and its auxiliary modules. Since this component is ATN program invariant, it is stored in a separate file and loaded into the compiled ATN machine when the ATN module component starts loading. In this section, we will examine the skeletons of the core component and the prototypes of ATN state and arc modules after we briefly discuss an issue about the control of search order.

7.3.1 Handling of side-effects and semi-flexible depth-first search

We have pointed out that, in the Kwasny-Shapiro interpreter, some SNePS structures may be created while a process is in execution, and that such a creation is a side-effect. In order to handle the side-effects inexpensively, the interpreter searches the ATN space in a depth-first fashion. We have imposed a non-trivial constraint in the compiled machine that restricts the degree of flexibility in the search order to a certain extent. In our implementation, we assume that the most recently created process's successors get the highest search priority and that the search order is flexible among the child processes of the most recently expanded processes, rather than among all the processes spread over the entire search frontier in the ATN net. This restriction appears to result in depth-first search. However, this is different from the *blind* depth-first control that the compiled ATN machines of the host-recursion model perform. We call the depth-first control of our kind, in which flexibility is allowed within each state, *semi-flexible* depth-first. Clearly, semi-flexible depth-first control, which is a kind of "hill-climbing strategy" [Winston (1984:93)], is mid-way between blind depth-first search and fully flexible search, and is still better than blind depth-first. The benefit of this constraint is that the side-effects caused by an arc module can be undone by its parent state module when the parent module regains control either to invoke another arc module or to find that there is no more left. However, by implementing a different procedure for handling side-effects, one can certainly build an ATN machine of this model to be fully flexible.¹⁹ Thus, this

¹⁹ Martins's (1983) implementation of multiple belief spaces in SNePS, in which the interpretation of SNePS structures are context dependent, suggests one possible solution to this problem.

implementational constraint should not be construed as a theoretical weakness of the model itself.

7.3.2 Control driver

The top-level control driver of the production-system ATN machine is the function PARSE as shown in Figure 4.

Figure 4. The top-level control driver
***** ABOUT HERE *****

PARSE takes a sequence of pre-processed input²⁰ and an atom for the name of the initial state as its arguments. It then initializes the process queue QCFGs with the given input and the initial state, and begins to crank the infinite loop of executing the frontmost process taken from the queue and updating the queue. As Figure 5 illustrates, EVAL-PC²¹ invokes the appropriate ATN state module with the process's configuration frame CFG as its argument.

Figure 5. EVAL-PC, a module for
a function application
***** ABOUT HERE *****

7.3.3 State modules

In this section, we give an example of compilation of an ATN state.²² A fragment of an actual ATN program as given in Figure 6 yields a compiled ATN module as shown in Figure 7. When entered, a state module first removes any side-effects left by the last-executed child arc alternative before it invokes the appropriate arc module pointed to by the entry point. Because of our policy of having the side-effects of the previous process removed by the process called for the next sibling process, we have implemented such that a state module is re-entered once

²⁰ The input pre-processing includes actions such as dictionary lookup, decapitalization, and punctuation handling.

²¹ EVAL-PC, as well as PARSE, is one of the best checkpoints to be monitored by ATN debuggers.

²² An ATN source state <state> is mapped to a compiled ATN state module with the name CA%<state>.

 Figure 6. A fragment of an ATN program
 ***** ABOUT HERE *****

 Figure 7. Compiled ATN state module
 corresponding to state SP
 ***** ABOUT HERE *****

more after all of its arc alternatives have been consumed, just for the purpose of removing the side-effects left by the last arc alternative.²³ If an entry value leads to a mis-firing of an arc module due to a rejection of the arc module's entrance test, the control may completely exit the state module and then re-enter the state module for the next arc alternative. However, in such a case in the present implementation, instead of exiting vacuously, the state module keeps invoking different arc modules, altering the entry values²⁴ until it finds the first successful one in order to cut down the frequency of re-entrance to a state module. An arc module returns a record with two information fields: a set of successor archiprocesses and a set of SNePS nodes created by the arc module as its side effect. However, if the entrance to the arc is rejected by the ATN's entrance test, a mere *NIL* report, instead of a record of two empty sets, is returned to the state module. This informative signaling method makes it possible for a state module to differentiate the case of an immediate test entrance failure from that of a blocked ATN search in the search space but leaving no side-effects and successor processes. This distinction is utilized by pseudo-state modules due to GROUP arcs in determining when the rest arc alternatives should be aborted. A state module returns to the top-level driver a set of processes (including possibly an empty set) which is the union of the newly expanded successor processes and the process itself to be re-entered.

7.3.4 Arc modules

²³ And, this is also a good time to give the user the message, if desired, that the control exits the state.

²⁴ Presently, the function NEXT-EN-EN that finds the next entry value, is a macro replaced by Lisp's ADD1 function. But, this macro is subject to be refined into any appropriate operation to meet the particular need of the individual installation site depending on their policy over search order.

Arc modules²⁵ in the compiled machine are closely related to the interpreter's arc functions. However, while the interpreter's arc functions are *generic*, the compiled machine's arc modules are *specific* to a state. A generic arc function takes, as its arguments, a specific ATN source of the arc body to be interpreted and a specific configuration frame of the current state, and interprets the arc body within the eval-frame as indicated by the configuration frame to realize all the tests and actions to be taken by the arc. But arc modules in the compiled machine are each bound to a state with some variable bindings and form expansions done during compile time. Therefore, a compiled arc module realizes the class of specific instances of an arc in a particular state, and is invoked in order to complete the last phase bindings and form evaluations that had to be delayed until parse time. Arc modules that realize an instance of CAT, TST, WRD, VIR, TO, or JUMP are straightforward in the sense that they are largely a mere transformation of the respective generic arc function filtered through the first-phase evaluation by the compiler. But the arc modules for the GROUP arc and push-family arcs deserve some comments on their somewhat tricky implementational tactics.

We pointed out that, in the interpreter machine (cf. Section 6.3), a list of the push-pop actions are stored in successor process's configuration frame in their ATN source notation so that it can be later interpreted. In the compiled machine (cf. Section 7.2), the push-pop actions are defined into a unique lambda function on the parser control's entrance to the associated PUSH process, and its address (i.e., the function name) is stored in the configuration frame so that it can be later invoked by the right POP process.

A GROUP arc makes itself as if it were a state. A set of arcs emanate from a GROUP arc very much like from a state. But the first positive result of the entrance test to any of the alternative arcs within a GROUP aborts all the other alternatives. We will call the imaginary

²⁵ The arc modules dependent on CA%<state> are named CA%<state><k><arcname-k> where <arcname-k> is the *k*-th arc emanating from the state <state> in the source program. This naming scheme systematically assigns every compiled ATN module a mnemonic name. However, there is a slim chance of having a collision between a state module and an arc module. For instance, if there is a state named NP with its, say, first arc being CAT, and if there is another state named NP1CAT, then the name CA%NP1CAT will be assigned both to the module for the state NP1CAT and to the module for the first arc CAT of the state NP. Collision of even this kind can be avoided by maintaining a table of module names, and letting it be checked before every new module is to be named, which, we believe however, will be a waste of resources with no practical danger existing.

state created by a GROUP arc a *pseudo-state*. A GROUP arc module²⁶ is by itself a pseudo-state module. It invokes its own arc modules with a control slightly different from that of normal state modules. Apparently, one of the strong motivations of having GROUP arc convention in ATN notation is to allow ATN programmers to express their special desire for a sequential control over otherwise non-deterministic alternatives. Thus, we also design GROUP arc modules such a way that the control flows sequentially just as indicated by the ATN programmer. For this reason, a pseudo-state made by a GROUP arc is an OR-construction to which the concept of re-entrant state module is not applied. Figure 8 exemplifies a fragment of an ATN program with a GROUP arc used in it, and Figure 9 illustrates how the GROUP arc is mapped to a pseudo-state module in a compiled ATN system.

 Figure 8. A fragment of an ATN program with a GROUP arc
 ***** ABOUT HERE *****

 Figure 9. Compiled ATN pseudo-state module
 corresponding to pseudo-state GROUP arc of state NP
 ***** ABOUT HERE *****

7.4 Parse time control

A clear advantage of the production-system model is that a vast amount of system control can be exercised at parse time. With an appropriate refinement of the algorithm for the entry point value computation at each entrance to a state module, the entirely different ATN search can be explored. Even without a sophisticated control strategy algorithmically set up, the user can experiment with different search controls manually using the convenient and powerful monitoring devices that can be easily inserted to the system because of the production-system architecture. The places that such monitoring devices can best fit are the two control driver modules, PARSE and EVAL-PC. PARSE is the module that drives the

²⁶ The pseudo state of a GROUP arc is represented by a "*" concatenated at the end of the name of the GROUP arc module such as CA%<state><k>GROUP*, and the embedded arcs from the pseudo state are further named consistently in accordance with the above rule for arc module naming.

production-system, and EVAL-PC is the module that fires the current processes, namely, each selected state module for every production-system cycle. For instance, by setting the global flag PARSE-MONITOR on, the user enters a break package every time the search driver iterates the production-system cycle as was illustrated in Figure 4. In this break package, the user can easily examine or manipulate the content of the process queue (QCFGs) in order to trace or modify the search order.²⁷ Similarly, by setting the global flag EVAL-PC-MONITOR on, the user enters a break package every time the search driver is about to execute the process selected from the process queue as was illustrated in Figure 5. The user can also enter the break loop when the control is about to invoke certain selected state modules only, by maintaining a global list BREAK-POINTS of state module names which is checked with for a break in EVAL-PC as shown in Figure 5. In the break package entered from EVAL-PC, the user can examine or manipulate the content of the current process (CFG). The user can freely and safely rearrange the order in which the state-dependent alternative arc modules are executed by merely altering the entry point value stored in the process's configuration frame. Figure 10 shows an example of a user exercising this control facility.

 Figure 10. A parse time trace of the control driver
 monitored by the user

***** ABOUT HERE *****

7.5 Results

Figure 11 illustrates an efficiency comparison between the two ATN machines, the interpreter and the compiled machines.²⁸

 Figure 11. Efficiency comparison between interpreted and
 compiled ATN machines with Shapiro (1985) ATN program

***** ABOUT HERE *****

²⁷ However, shuffling of the processes in this queue should be done with a great amount of caution exercised if the site-dependent policy of side-effects removal imposes a certain limitation.

²⁸ We made this comparison with all the other factors remaining precisely the same. The two Lisp programs were both Lisp compiled for their maximum speed, and the test run was made when there was no other users logged in on the system.

The ATN program used was the grammar for natural language processing by Shapiro (1985). Some inputs, prefixed by an "S", were English sentences which caused the system to respond in English through the whole chain of processings for surface sentence understanding, inference, and surface sentence generation, while some others, prefixed with "G", were SNePS structures which caused the system to generate surface English sentences expressions for them. The input forms were given to the system in the order as they are presented in the figure, which caused the system to construct and maintain a particular cognitive context. With these non-trivial data, the compiled machine was found to be running about 3 times as fast as the interpreter counterpart. Considering the factor that this comparison was made in SNePS context where many SNePS activities other than ATN parsing *per se*, such as deductions, were also taking place embedded in parsing routines, we reasonably conclude that the actual efficiency improvement should be far more than this figure. Through further innumerable successive tests carried out in SNePS Group Lab, we were convinced that the presented design of our compiled machine contained no apparent faults.

8. CONCLUSION

Compiled ATN machines of the production-system model constructed along the incremental compilation approach are important. We have demonstrated that the design technique of re-entrant state module, in which an ATN state module is built so as to contain as many entry points as the number of the arcs in it, is a valid solution to the problem of building a compiled production-system ATN machine that allows flexible search control. This design provides enough motivation to go further into arc-module modularization beyond the traditional state-module modularization, thus extending the scope of incremental compilation of ATN programs. Our satisfactory implementation has proven that this design is practically viable, too, by enhancing the efficiency of ATN program execution in terms of the speed as much as three times compared with the interpreter machine in the same environment. The practical benefit of flexible search control an ATN machine of this design has hinges upon the effectiveness of

the algorithm used to calculate the value for the entry-point on each entrance to a state module. The study of this issue remains open.

REFERENCES

- Bates, M. (1978) "The theory and practice of augmented transition network grammars". In L. Bolc (ed.), *Natural Language Communication with Computers*: 191-259. Spring Verlag, Berlin.
- Bobrow, D. G. & Fraser (1969) "An Augmented State Transition Network Analysis Procedure", in *IJCAI-1969*.
- Bobrow, D. G., Kaplan, R. M., Kay, M., & Norman, D. A., Thompson, H. & Winograd, T. (1977) "GUS: a Frame-Driven Dialogue System", in *Artificial Intelligence* 8: 155-173.
- Burton, R. R. (1976) "Semantic grammar: An engineering technique for constructing natural language understanding systems". BBN Report No. 3453, Cambridge, Mass.
- Burton, R. R. & Woods, W. A. (1976) "A compiling system for augmented transition networks". Preprints of 6. COLING at Univ. of Ottawa.
- Chomsky, N. & Halle, M. (1968) *The Sound Pattern of English*. Harper & Row, Publishers, New York.
- Chrastaller, T. (1983) "An ATN programming environment". In L. Bolc (ed.), *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*. Spring-Verlag, New York.
- Davis, R. & King, J. J. (1977) "An overview of production systems". In E. Elcock and D. Michie (eds.), *Machine intelligence* 8: 300-332. Ellis Horwood, Chichester, England.
- Finin, T. W. (1977) "An interpreter and compiler for augmented transition networks". Report T-48, Coordinated Science Laboratory, University of Illinois.
- Finin, T. W. (1983) "The planes interpreter and compiler for augmented transition network grammars". In L. Bolc (ed.), *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*. Spring-Verlag, New York.
- Kaplan, R. M. (1972) "Augmented transition networks as psychological models of sentence comprehension". *Artificial Intelligence* 3: 77-100.
- Kaplan, R. M. (1973a) "A general syntactic processor". In R. Rustin (ed.), *Natural Language Processing*: 193-241.
- Kaplan, R. M. (1973b) "A multi-processing approach to natural language". *Proc. of the 1973 National Computer Conference*: 435-440, at Montvale, NJ.
- Kaplan, R. M. (1975) "On process models for sentence analysis". In D. Norman, D. Rumelhart & the LNR Research Group (eds.), *Explorations in Cognition*: 117-135. Freeman, San Francisco, CA.
- Kochut, K. (1983) "Towards the elastic ATN implementation". In L. Bolc (ed.), *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*. Spring-Verlag, New York.
- Kwasny, S. (1974) "Implementation of an ATN parsing system". Unpublished manuscript at Computer Science Dept., Indiana Univ., August, 1974.
- Kwasny, S. C. & Sondheimer, N. K. (1981) "Relaxation techniques for parsing ill-formed input". *AJCL*: 7: 99-108.
- Laubsch, J. & Barth, K. (1983) "Compiling augmented transition networks into MacLisp". In L. Bolc (ed.), *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*. Spring-Verlag, New York.
- McKay, D. P. & Martins, J. P. (1981) "SNePSLOG user's manual". SNeRG Tech. Note 4. Computer Science Dept., State Univ. of New York at Buffalo.

- Martins, J. P. (1983) "Reasoning in multiple belief spaces". Doctoral dissertation at State Univ. of New York at Buffalo. Tech. Rep. 203, Computer Science Dept., State Univ. of New York at Buffalo.
- Post, E. (1943) "Formal reductions of the general combinatorial problem". *American Journal of Mathematics* 65: 197-268.
- Shapiro, S. C. (1975) "Generation as parsing from a network into a linear string". *AJCL Microfiche* 33: 45-62.
- Shapiro, S. C. (1979) "The SNePS semantic network processing system". In N. V. Findler (ed.), *Associative Networks: the Representation and Use of Knowledge by Computers*. 12-25. New York: Academic Press.
- Shapiro, S. C. (1982) "Generalized augmented transition network grammars for generation from semantic networks". *AJCL*-8: 12-25.
- Shapiro, S. C. & Rapaport, W. J. (1985) "SNePS considered as a fully intensional propositional semantic network", Tech. Rep. 85-15, Computer Science Dept., State Univ. of New York at Buffalo. (Also forthcoming in G. McCalla & N. Cercone (eds.) *Knowledge Representation*. Springer-Verlag, Berlin).
- Tennant, H. (1981) *Natural Language Processing*. PBI-Petrocelli Books. Princeton, NJ.
- Thorne, J., Bratley, P. & Dewar, H. (1968) "The syntactic analysis of English by machine". *Machine Intelligence* 3. Elsevier, New York.
- Waltz (1976) "The PLANES system: natural language access to a large data base", C.S.I. Tech. Rep. T-34.
- Wanner, E. & Maratsos, M. (1978) "An ATN approach to comprehension". In M. Halle, J. W. Bresnan & G. A. Miller (eds.), *Linguistic Theory and Psychological Reality*: 119-161. MIT Press, Cambridge, Mass.
- Weischedel, R. M. & Sondheimer, N. K. (1984) "Meta-rules as a basis for processing ill-formed output". *AJCL*-9: 161-177.
- Winograd, T. (1983) *Language as a Cognitive Process - Volume 1: Syntax*. Addison-Wesley Pub., Reading, Mass.
- Winston, P. H. (1984) *Artificial Intelligence* (2nd ed.). Addison-Wesley Pub., Reading, Mass.
- Woods, W. A. (1970) "Transition network grammars for natural language analysis". *CACM*-13: 591-606.
- Woods, W. A. (1973) "An experimental parsing system for transition network grammars". In R. Rustin (ed.) *Natural Language Processing*: 111-154. Algorithmics Press, New York.
- Woods, W. A. (1978) "Research in natural language understanding". BBN Report No. 3963.
- Woods, W. A. (1980) "Cascaded ATN grammars". *AJCL*-6: 1-18.
- Woods, W. A., Bates, M., Brown, G., Bruce, B., Cook, C., Klovstad, J., Makhoul, J., Nash-Webber, B., Schwartz, R., Wolf, J. & Zue, V. (1976) "Speech understanding systems - Final Technical Report". Tech Rep. 3438. BBN, Cambridge, MASS.
- Woods, W. A., Kaplan, R. M. & Nash-Webber, B. (1972) "The Lunar Science Language System: Final Report". Tech Rep 2378. BBN, Cambridge, Mass.

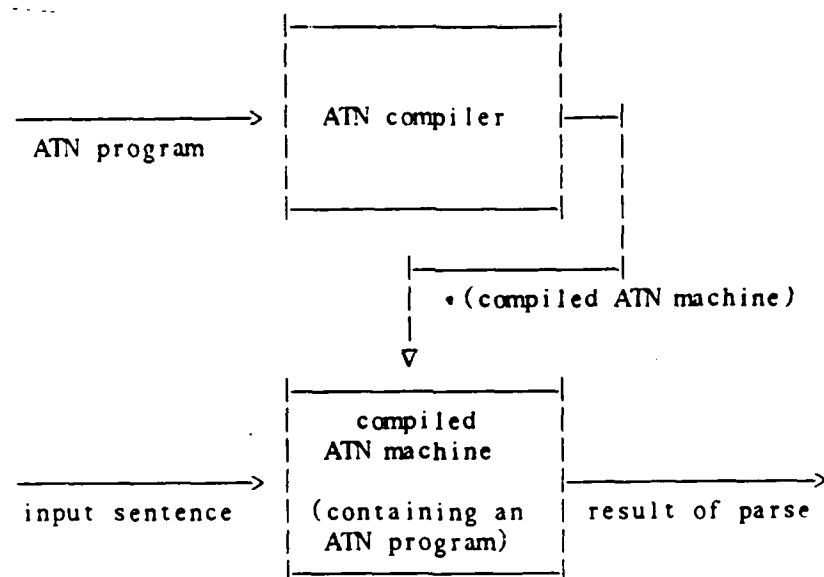
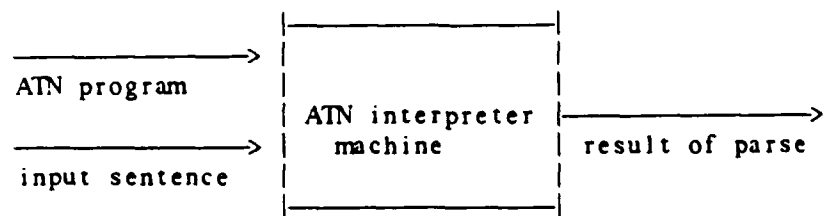


Figure 1. Input/output relations of various ATN-related machines

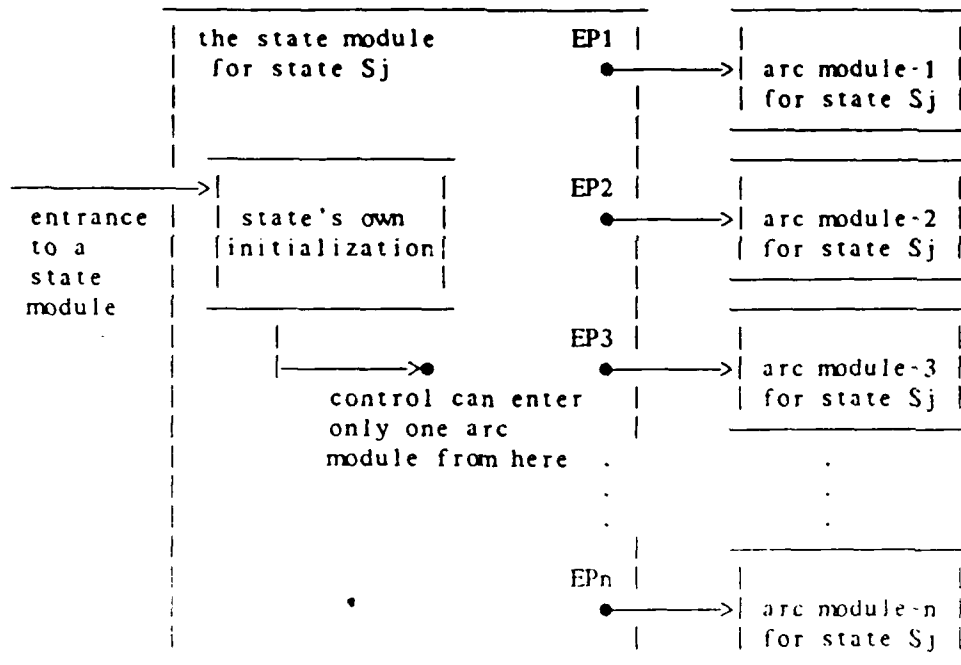


Figure 2. Block-diagram of the control flow through a re-entrant state module to an arc module

```

(DEF COMPARC
  (LAMBDA (OPORT SSID ACCOUNT ARC)
    ;; OPORT is the compiler output file
    ;; SSID is the name of the state which the arc is dependent on
    ;; ACCOUNT counts the static order of the arc in the state
    ;; ARC is the source ATN definition of the arc
    (LET ((AMODULE-NAME (MAKE-AMODULE-NAME SSID ACCOUNT ARC)))
      (COND ((IS-VALID-ARC ARC)
              (SETQ CA-ARCS (CONS AMODULE-NAME CA-ARCS))
              ;; a global variable, CA-ARCS remembers all the arc modules generated
              (FUNCALL (GET-COMPARC-SPECIALIST ARC)
                       OPORT
                       AMODULE-NAME
                       ARC))
            (T (PRINT-CODE OPORT ";;**ERROR IN ARC FUNCTION**")
                (MSG N "UNKNOWN ARC " (ARC-NAME ARC) " IN " SSID)
                NIL))))))

```

Figure 3. Lisp definition of COMPARC function

```

PROCEDURE COMPARC (OUTFILE; STATE-NAME; ARC-COUNT; ARC-BODY);
{
  generate the name of the compiled arc module;
  IF no error in ARC-BODY
    THEN
      call arc-specialist to generate code for the compile arc module;
    ELSE
      give message for an error in ARC-BODY of STATE-NAME;
      write in OUTFILE an error-indicating code;
}

```

Figure 3. Pseudo-code description of COMPARC

```

(DEF PARSE
  (LAMBDA (PREP-SENTENCE INIT-STATE)
    ;; PREP-SENTENCE is the pre-processed input sentence
    ;; INIT-STATE is the name of the ATN state to start from
    (DO ((QCFG (INITIALIZE-PCQ PREP-SENTENCE INIT-STATE))
        (PARSE-SET NIL))
      ; PARSE-SET is non-locally modified by successful parsings
      ((ISNULL-PCQ QCFG) PARSE-SET)
      ; if the queue is empty, returns PARSE-SET
      (BREAK PARSE-BREAK (AND PARSE-MONITOR (EXAM-QUEUE)))
      ; if PARSE-MONITOR flag is on,
      ; displays the process queue, and enters the break package
      (SETQ QCFG
        (MERGE-PCQ (EVAL-PC (FIRST-PC!PCQ QCFG))
          (REST-PCQ QCFG)))
      ; the queue is updated with the result of the current process
      (IF (AND ONE-PARSEFLG PARSE-SET) (SETQ QCFG NIL-PCQ))))

```

Figure 4. Top-level control driver

```

(DEF EVAL-PC
  (LAMBDA (CFG)
    ;; CFG is the configuration frame of the current process
    (BREAK EVAL-PC-BREAK (AND EVAL-PC-MONITOR (EXAM-PC)))
    ; if EVAL-PC-MONITOR flag is on,
    ; displays the config. frame, and enters the break package
    (*BREAK (IS-BREAKPOINT-SET (GET-STATE-MODULE-NAME CFG) BREAK-POINTS)
      (CONS 'break-at-bpoint (GET-STATE-MODULE-NAME CFG)))
    ; enters the break package
    ; if the state module name is listed in BREAK-POINTS
    (FUNCALL (GET-STATE-MODULE-NAME CFG) CFG))

```

Figure 5. EVAL-PC, a module for
a function application

```

(SP ; parse a sentence
  (WRD (WHO WHAT) T
    ; if it starts with "who" or "what", it's a question.
    (SETR TYPE 'Q) (LIFTR TYPE) (SETR SUBJ %X) (TO V))
  (PUSH NPP T
    ; a statement starts with a noun phrase — its subject.
    (SEDR TYPE 'D) (SEDR P-ATT) (SEDR NH2)
    (SEDR BELIEVER.REG)
    (SETR TYPE (COND ((NULLR STYPE) 'D)
                     (T '(D S))))
    (LIFTR TYPE) (SETR SUBJ *)
    ....(TO V)))

```

Figure 6. A fragment of an ATN program
(Note that there are two arc alternatives in state SP)

```

(DEF CA%SP
  (LAMBDA (CFG)
    (CLEAN-SIDEEFFS (GET-SIDEEFFECTS CFG))
    ; First, the sideeffects left by the previous entrance are undone
    (FIX) ; DO-loop keeps the control to stay in the state module
    ; until the first successful arc alternative is found.
    (ENTRY (GET-EN!PC CFG))
    ; ENTRY is the value of the entry-point
    (NDS+SONS))
    ; the value returned from the successor arc module
    (NIL 'INFINITE-LOOP)
    (SETQ NDS+SONS
      (CASEQ ENTRY
        (1 (CA%SP1WRD CFG))
        (2 (CA%SP2PUSH CFG))
        (T (RETURN NIL-PCQ))))
    (SETQ ENTRY (NEXT-EN.EN ENTRY))
    (COND (NDS+SON
      (RETURN (MERGE-PCQ.PCQ&PC
        (SUCCESSOR-PCQ NDS+SONS)
        (UPDATE-CURRENT-PC
          CFG
          (GET-SIDEEFFECTS NDS+SONS)
          (NEXT-EN.EN ENTRY))))))
      (T 'DO-LOOPING))))

```

Figure 7. Compiled ATN state module corresponding to state SP
(Note that the compiled state module CA%SP for state SP
contains in it two non-trivial entry-points)

```

(NP
  (GROUP ; first see if there's a propositional attitude
    (PUSH G (OR (GETA WHICH) (GETA VERB))
      ; generate an S to express the M-objective
      ; of a propositional attitude if there is one.
      ; if the OBJ-register contains a propositional node.
      ; then that node has either a which-arc or a verb-arc
      (SENR DONE) (SENR DUMMY (RTPRINT 'THAT)) (TO END))
    ;else: the proper number is pl for a class, sing for an individual
    (JUMP NP1 NUMBR)
    (JUMP NP1 (OR (GETA SUB-) (GETA SUP-) (GETA CLASS-))
      (SETR NUMBR 'PL))
    (JUMP NP1 (NOT (OR (GETA SUB-)(GETA SUP-)(GETA CLASS-)))
      (SETR NUMBR 'SING))))

```

Figure 8. A fragment of an ATN program with a group arc
(Note that the GROUP arc groups the four sub-arcs in it
into a sequence of mutually exclusive alternatives.)

```

(DEF CA%NP1GROUP
  (LAMBDA (CFG)
    (CLEAN-SIDEFFS (NDS!PC CFG))
    (OR (CA%NP1GROUP*1PUSH CFG)
      (CA%NP1GROUP*2JUMP CFG)
      (CA%NP1GROUP*3JUMP CFG)
      (CA%NP1GROUP*4JUMP CFG))))

```

Figure 9. Compiled ATN pseudo-state module
corresponding to pseudo-state group arc of state NP
(Note that the four sub-arc modules are combined
into a sequential OR-construction.)

[PARSE-MONITOR IS ON]

: Young Lucy petted a yellow dog

Process Queue

1: (1 nil (young & Lucy & ...) CA%*s* ...)

Break parse-break

<1>: (return t)

Process Queue

1: (1 nil (young & Lucy & ...) CA%*sp* ...)

2: (2 nil (young & Lucy & ...) CA%*s* ...)

Break parse-break

<1>: (setq parse-monitor nil eval-pc-monitor t)

t

[NOW, PARSE-MONITOR IS OFF, AND EVAL-PC-MONITOR IS ON]

<1>: (return t)

Configuration Frame for (CA%*sp* . 1)

(entry = 1)

(nodeset = NIL)

(buffer = (young (&) Lucy (&) ...))

(state = CA%*sp*)

(pstack = ((CA%*s* & CA%*s*1push-pop2)))

(level = 2)

(regs = ((believer.reg . Cassie)))

(held = NIL)

(path = NIL)

(weight = 0)

Break eval-pc-break

<1>: (setq eval-pc-monitor nil)

nil

[NOW, EVAL-PC-MONITOR IS ALSO OFF]

<1>: (set-bpoint npdet)

A break-point made at state-module CA%*npdet*

[NOW, A BREAK-POINT IS SET TO THE STATE MODULE FOR STATE npdet]

CA%*npdet*

<1>: (return t)

Break (break-at-bpoint . CA%*npdet*)

<1>: (exam-queue)

Process Queue

1: (1 nil (young & Lucy & ...) CA%*npdet* ...)

2: (5 nil (young & Lucy & ...) CA%*np* ...)

3: (3 nil (young & Lucy & ...) CA%*clause* ...)

4: (11 nil (young & Lucy & ...) CA%*rulep* ...)

5: (2 nil (young & Lucy & ...) CA%*rpl* ...)

6: (2 nil (young & Lucy & ...) CA%*sp* ...)

7: (2 nil (young & Lucy & ...) CA%*s* ...)

t

<1>: (return t)

```

Break (break-at-bpoint . CA%npdet)
<1>: (exam-queue)
Process Queue
1: (1 nil (Lucy & petted & ...) CA%npdet ...)
2: (2 nil (young & Lucy & ...) CA%npdet ...)
3: (5 nil (young & Lucy & ...) CA%npdet ...)
4: (3 nil (young & Lucy & ...) CA%npdet ...)
5: (11 nil (young & Lucy & ...) CA%npdet ...)
6: (2 nil (young & Lucy & ...) CA%npdet ...)
7: (2 nil (young & Lucy & ...) CA%npdet ...)
8: (2 nil (young & Lucy & ...) CA%npdet ...)
t
<1>: (exam-pc)
Configuration Frame for (CA%npdet . 1)
(entry = 1)
(nodeset = NIL)
(buffer = (Lucy (&) petted (& &) ...))
(state = CA%npdet)
(pstack = ((CA%npdet & CA%npdet2push-pop1) (CA%npdet & CA%npdet10push-pop2) ...))
(level = 5)
(regs = ((* . young) (* . young) (lex . young) (* . young) ...))
(held = ((adj 5 & nil)))
(path = NIL)
(weight = 0)
t
<1>: (reset-entry 6)
entry value is set to 6
6
[ NOW, ENTRY VALUE IS ALTERED FROM 1 TO 6 ]
<1>: (exam-pc)
Configuration Frame for (CA%npdet . 6)
(entry = 6)
(nodeset = NIL)
(buffer = (Lucy (&) petted (& &) ...))
(state = CA%npdet)
(pstack = ((CA%npdet & CA%npdet2push-pop1) (CA%npdet & CA%npdet10push-pop2) ...))
(level = 5)
(regs = ((* . young) (* . young) (lex . young) (* . young) ...))
(held = ((adj 5 & nil)))
(path = NIL)
(weight = 0)
t
<1>: (unset-bpoint npdet)
The break-point removed from state-module CA%npdet
CA%npdet
[ THE BREAK-POINT AT STATE MODULE CA%npdet IS REMOVED ]
<1>: (return t)
I understand that young Lucy petted a yellow dog

(time (sec.) : => (cpu= 5.733 gc= 0.00) <=)

```

Figure 10. A parse time trace of the control driver

Parse tasks

Usage of symbols for task identification

S* (Surface to surface parsing) :
input sentence → system's response
G* (Semantic structure to surface parsing) :
>> system's expression

S1: young Lucy petted a yellow dog
→ I understand that young Lucy petted a yellow dog
G1: >> young Lucy will pet a yellow dog
G2: >> young Lucy is starting to pet a yellow dog
G3: >> young Lucy is petting a yellow dog
G4: >> young Lucy just petted a yellow dog
G5: >> young Lucy petted a yellow dog
G6: >> young Lucy will be petting a yellow dog
G7: >> young Lucy was petting a yellow dog
G8: >> young Lucy would pet a yellow dog
G9: >> young Lucy will have petted a yellow dog
S2: what is yellow → a dog is yellow
S3: dogs are animals → I understand that dogs are animals
S4: who petted an animal → young Lucy petted a yellow animal
S5: Lucy is a girl
→ I understand that sweet young Lucy is a girl
S6: girls are people → I understand that girls are people
S7: what was petted by a person
→ a yellow animal was petted by sweet young Lucy
S8: who petted a dog
→ sweet young Lucy petted a yellow animal
G10: >> a yellow animal
S9: who petted Rover → sweet young Lucy petted yellow Rover

task	interpreted	compiled	task	interpreted	compiled
S1	16.03	2.93	G5	5.45	1.73
S2	13.65	4.53	S7	30.66	10.18
G1	5.81	1.81	G6	7.50	2.38
S3	4.71	1.53	S8	30.73	10.21
G2	5.06	1.58	G7	6.93	2.16
S4	30.06	9.93	G10	2.58	0.88
G3	6.51	2.05	G8	13.43	4.28
S5	7.18	2.26	S9	19.96	6.65
G4	5.13	1.65	G9	11.95	3.91
S6	4.68	1.50			
Ratio	3.160 : 1 (or 1 : 0.316)				

Figure 11. Efficiency comparison between interpreted and compiled ATN machines with Shapiro (1985) ATN program (time unit = CPU seconds)

Appendix B4:

"SNePS Considered as a Fully Intensional Propositional Semantic Network"

Shapiro, S.C.
Rapaport, R.J.

11.1. Introduction

This chapter presents a formal syntax and semantics for SNePS, the *Semantic Network Processing System* (Shapiro, 1979b).³⁰ The syntax shows the emphasis placed on SNePS's *propositional* nature. The semantics, which is based on Alexius Meinong's theory of intentional objects (the objects of thought), makes SNePS's *fully intensional* nature precise: as a fully intensional theory, it avoids possible worlds and is appropriate for AI considered as "computational philosophy" - AI as the study of how intelligence is possible - or "computational psychology" - AI with the goal of writing programs as models of *human* cognitive behavior. We also present a number of recent AI research and applications projects that use SNePS, concentrating on one of these, a use of SNePS to model (or construct) the mind of a cognitive agent, referred to as CASSIE (the Cognitive Agent of the SNePS System-an Intelligent Entity).

11.1.1. The SNePS environment

A semantic network is a data structure typically consisting of labeled nodes and labeled, directed arcs. SNePS can be viewed as a semantic network language with facilities for

1. building semantic networks to represent virtually

³⁰This research was supported in part by the National Science Foundation under Grant No. IST-8504713 and SUNY Buffalo Research Development Fund grants No. 150-9216-F and No. 150-8537-G (Rapaport), and in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, NY 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-85-C-0008 (Shapiro). We wish to thank Michael Almeida, James Geller, João Martins, Jeannette Neal, Sargur N. Srihari, Jennifer Suchin, and Zhigang Xiang for supplying us with descriptions of their projects, and Randall R. Dipert, the members of SNeRG (the SNePS Research Group), and three anonymous reviewers for comments and discussion.

any kind of information or knowledge.

2. retrieving information from them, and
3. performing inference with them, using SNIP (the *SNePS Inference Package*) and path-based inference.

Users can interact with SNePS in a variety of interface languages, including: SNePSUL, a LISP-like SNePS User Language; SENECA, a menu-based, screen-oriented editor; GINSENG, a graphics-oriented editor; SNePSLOG, a higher-order-logic language (in the sense in which PROLOG is a first-order-logic language) (McKay and Martins, 1981). (Shapiro, McKay, Martins, and Morgado, 1981); and an extendible fragment of English, using an ATN parsing and generating grammar (Shapiro, 1982), see Figure 11-1.

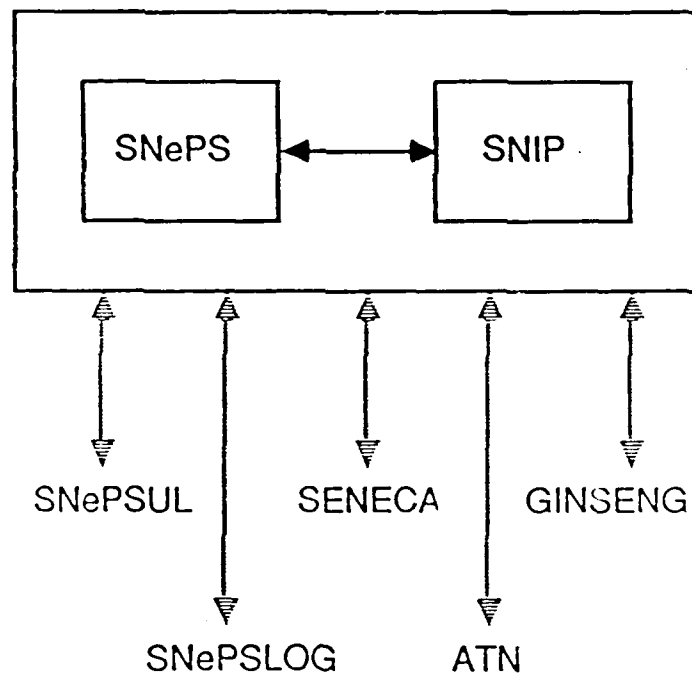


Figure 11-1: SNePS, SNIP and Their User Interfaces.

SNePS, SNIP (the SNePS Interface Package) and their user interfaces. When the arcs, case frames, and ATN grammar are those of SNePS/CASSIE, then the system is being used to model CASSIE. When the arcs are for the database (see Section 11.4.1), then the system is being used as a database management system, etc.

SNePS is the descendent of SAMENLAQ (Shapiro, Woodmansee, and Kreuger, 1968) (Shapiro and Woodmansee, 1969) and MENTAL (Shapiro, 1971b), (Shapiro, 1971c). It was developed with the help of the SNePS Research Group at Indiana University and at the University at Buffalo. The current version is implemented in Franz LISP and runs on VAX 11/750s and 780s in the Department of Computer Science at Buffalo. An earlier version was implemented in ALISP on a CDC Cyber 730; and an updated version is being implemented in Common LISP on Symbolics LISP machines, TI Explorers, and a Tektronix 4406. There are additional installations at other universities in the U.S. and Europe.

11.1.2. SNePS as a knowledge representation system

Some researchers, for example, (Levesque and Brachman, 1985), view a knowledge representation (KR) system as a subsystem that manages the knowledge base of a knowledge-based system by storing information and answering questions. In contrast, we view SNePS as the entire knowledge-based system, interacting with a user/interlocutor through one of its interfaces. Of course, the user/interlocutor could be another computer program using SNePS as a subsystem, but that is not the way we use it.

A basic design goal of SNePS and its ancestors was to be an environment within which KR experiments could be performed, that is, to be a semantic network at the "logical" level, to use Brachman's term (Brachman, 1979), see Section 11.5, below. This has been effected by providing a rather low level interface, SNePSUL. Using SNePSUL, a KR designer can specify a syntax: individual arc labels and sets of arc labels (or case frames) that will be used to represent various objects and information about them. It is also the designer's obligation to supply a semantics for these case frames. As is the case for any provider of a language or "shell", we cannot be responsible for what use others make of the facilities we provide. Nevertheless, we have our own preferred use.

In this chapter, we try to do two things. First, we try to provide an understanding of SNePS and of some of the uses

to which it has been put. Second, and most importantly, we present our own preferred use: this is to use SNePS, with a particular set of arc labels and case frames, and a particular parsing/generating grammar for a fragment of English, as (a model of) the cognitive agent, CASSIE. We shall refer to SNePS with these arcs, case frames, and grammar as SNePS/CASSIE. SNePS/CASSIE forms CASSIE's mind and stands as our current theory of KR at the "conceptual" level (cf. Section 11.5, below, and (Brachman, 1979)). The purpose of the central part of this paper is to present this theory by explaining the entities represented by structures in SNePS/CASSIE, by giving a formal syntax and semantics for those structures, and by showing and explaining a sample conversation with CASSIE.

11.1.3. Informal description of SNePS

Regardless of the intentions of a KR-system designer, SNePS, as a KR formalism, provides certain facilities and has certain restrictions. The facilities (for example, for building, finding, and deducing nodes) are best understood as those provided by SNePSUL, but we shall not give a complete description of SNePSUL here. [For an example, cf. Section 11.4.1, below; for details, see (Shapiro, 1979b).] The restrictions, however, are important to understand, because they distinguish SNePS from a general labelled, directed graph and from many other semantic network formalisms.

SNePS is a *propositional* semantic network. By this is meant that all information, including propositions, "facts", etc., is represented by nodes. The benefit of representing propositions by nodes is that propositions about propositions can be represented with no limit. (In the formal syntax and semantics given in Section 11.3, the propositions are the nodes labelled 'm' or 'r'.)

Arcs merely form the underlying syntactic structure of SNePS. This is embodied in the restriction that one cannot add an arc between two existing nodes. That would be tantamount to telling SNePS a proposition that is not represented as a node. There are a few built-in arc labels, used mostly for

rule nodes. *Paths* of arcs can also be defined, allowing for *path-based* inference, including property inheritance within generalization hierarchies [see Section 11.3.4. below; cf. Shapiro (Shapiro, 1978), (Srihari, 1981), and (Tranch, 1982).] All other arc labels are defined by the user, typically at the beginning of an interaction with SNePS, although new labels can be defined at any time.

For purposes of reasoning, propositions that are asserted in SNePS must be distinguished from those propositions that are merely represented in SNePS but not asserted. This could happen in the case of a proposition embedded in another (for example, "Lucy is rich" embedded in "John believes that Lucy is rich"). SNePS interprets a proposition node to be asserted if and only if it has no arcs pointing to it.³¹

Another restriction is the *Uniqueness Principle*: There is a one-to-one correspondence between nodes and represented concepts. This principle guarantees that nodes will be shared whenever possible and that nodes represent intensional objects.³² We next consider the nature of these objects.

11.2. Intensional knowledge representation

SNePS can be used to represent propositions about entities in the world having properties and standing in relations. Roughly, nodes represent the propositions, entities, properties, and relations, while the arcs represent structural links between these.

SNePS nodes *might* represent *extensional* entities. Roughly, extensional entities are those whose "identity conditions" (the conditions for deciding when "two" of them are really the "same") do not depend on their manner of representation. They

³¹This is not really a restriction of SNePS, but of SNIP (the SNePS Inference Package) and path-based inference.

³²In (Maida and Shapiro, 1982) this name was given to only half of the Uniqueness Principle as stated here: "each concept represented in the network is represented by a unique node" (page 291).

may be characterized as those entities satisfying the following rough principle:

Two extensional entities are equivalent (for some purpose) if and only if they are identical³³

For example, the following are extensional:

- the Fregean referent of an expression;
- physical objects;
- sentences;
- truth values;
- mathematical objects such as:
 - sets,
 - functions defined in terms of their input-output behavior (that is, as sets of ordered pairs),
 - n -place relations defined in terms of sets of ordered n -tuples.

Although SNePS *can* be used to represent extensional entities in the world, we believe that it *must* represent *intensional* entities. Roughly, intensional entities are those whose identity conditions *do* depend on their manner of representation. They are those entities that satisfy the following rough principle:

Two intensional entities might be equivalent (for some) purpose without being identical (that is, they might really be two, not one).

Alternatively, intensional entities may be characterized as satisfying the following five criteria:

1. They are non-substitutable in referentially opaque contexts.
2. They can be indeterminate with respect to some properties.
3. They need not exist.

³³that is, if and only if "they" are really one entity, not two

4. They need not be possible.
5. They can be distinguished even if they are necessarily identical (for example, *the sum of 2 and 2* and *the sum of 3 and 1* are distinct objects of thought).

For example, the following are intensional:

- the Fregean sense of an expression;
- concepts;
- propositions;
- properties;
- algorithms;
- objects of thought, including:
 - fictional entities (such as Sherlock Holmes),
 - non-existents (such as the golden mountain),
 - impossible objects (such as the round square)

Only if one wants to represent the relations between a mind and the world would SNePS also have to represent extensional entities [cf. (Rapaport, 1976), (Rapaport, 1978), (McCarthy, 1979)]. However, if SNePS is used just to represent a mind - that is, a mind's model of the world-then *it does not need to represent any extensional objects*. SNePS can then be used either to model the mind of a particular cognitive agent or to build such a mind - that is, to *be* a cognitive agent itself.

There have been a number of arguments presented in both the AI and philosophical literature in the past few years for the need for intensional entities. (Castaneda, 1974), (Woods, 1975), (Rapaport, 1976), (Rapaport, 1985a), (Brachman, 1977), (Routley, 1979), cf. (Rapaport, 1984a), (Parsons, 1980), cf. (Rapaport, 1985b)). Among them, the following considerations seem to us to be especially significant:

Principle of Fine-Grained Representation:

The objects of thought (that is, intensional objects) are intensional: a mind can have two or more objects of thought that correspond to only one extensional object.

To take the classic example, the Morning Star and the Evening Star might be distinct objects of thought, yet there is only one

extensional object (viz., a certain astronomical body) corresponding to them.

Principle of Displacement:

Cognitive agents can think and talk about non-existents:
a mind can have an object of thought that corresponds to
no extensional object.

Again to take several classic examples, cognitive agents can think and talk about fictional objects such as Santa Claus, possible but non-existing objects such as a golden mountain, impossible objects such as a round square, and possible but not-yet-proven-to-exist objects such as theoretical entities (for example, black holes).

If nodes only represent intensional entities (and extensional entities are not represented in the network), how do they link up to the external, extensional world? In SNePS/CASSIE, the answer is by means of a LEX arc (see syntactic formation rule SR.1 and semantic interpretation rule SI.1 in Section 11.3.3, below): the nodes at the head of the LEX arc are *our* (the user's) interpretation of the node at its tail. The network without the LEX arcs and their head-nodes displays the *structure* of CASSIE's mind [cf. (Carnap, 1967), Section 11.14].

A second way that nodes can be linked to the world is by means of sensors and effectors, either linguistic or robotic. The robotic sort has been discussed in (Maida and Shapiro, 1982). Since so many AI understanding systems deal exclusively with language, here we consider a system with a keyboard as its sense organ and a CRT screen as its only effector.

Since the language system interacts with the outside world only through language, the only questions we can consider about the connections of its concepts with reality are questions such as:

Does it use words as we do?
When it uses word *w*, does it mean the same thing as
when I use it?
When I use word *w*, does it understand what I mean?

The perceptual system of the language system is its parser/analyzer - the programs that analyze typed utterances and build pieces of semantic network. The motor system is the generator - the programs that analyze a section of the semantic network and construct an utterance to be displayed on the CRT. One crucial requirement for an adequate connection with the world is simple consistency of input-output behavior. That is, a phrase that is analyzed to refer to a particular node should consistently refer to that node, at least while there is no change in the network. Similarly, if the system generates a certain phrase to describe the concept represented by a node, it should be capable of generating that same phrase for that same node, as long as nothing in the network changes. Notice that it is unreasonable to require that if a phrase is generated to describe a node, the analyzer should be able to find the node from the phrase. The system might know of several brown dogs and describe one as "a brown dog"; it could not be expected to find that node as the representation of "a brown dog" consistently.

If we are assured of the simple input-output consistency of the system, the main question left is whether it uses words to mean the same thing as we do. It is the same question that we would be concerned with if we were talking with a blind invalid, although in that case we would assume the answer was 'Yes' until the conversation grew so bizzare that we were forced to change our minds. As the system (or the invalid) uttered more and more sentences using a particular word or phrase, we would become more and more convinced that it meant what we would mean by it, or that it meant what we might have described with a different word or phrase ("Oh! When you say 'conceptual dependency structure', you mean what I mean when I say 'semantic network'."), or else that we *didn't* know what was meant, or that it was not using it in a consistent, meaningful way (and hence that the system (or invalid) did not know what it was talking about). As long as the conversation proceeds without our getting into the latter situation, the system has all the connections with reality it needs.

11.3. Description of SNePS/CASSIE

In this section, we introduce CASSIE, and give the syntax and semantics for SNePS/CASSIE in terms of a philosophical theory of mental entities inspired by Alexius Meinong's Theory of Objects.

11.3.1. CASSIE - A model of a mind

SNePS nodes represent the objects of CASSIE's thoughts - the things she thinks about, the properties and relations with which she characterizes them, her beliefs, her judgments, etc. [cf. (Maida and Shapiro, 1982), (Rapaport, 1985a)]. According to the Principle of Displacement, a cognitive agent is able to think about virtually anything, including fictional objects, possible but non-existing objects, and impossible objects. Any theory that would account for this fact requires a non-standard logic, and its semantics cannot be limited to merely *possible* worlds. (Otherwise, it could not account for impossible objects. This accounts for the difficulties David Israel has in providing a possible-worlds semantics for SNePS (Israel, 1983), (cf. (Rapaport, 1985a)). Theories based on the Theory of Objects of the turn-of-the-century Austrian philosopher-psychologist Alexius Meinong are of precisely this kind.

For present purposes, it will be enough to say that Meinong held that psychological experiences consist in part of a psychological *act* (such as thinking, believing, judging, wishing, etc.) and the *object* to which the act is directed (for example, the object that is thought about or the proposition that is believed). Two kinds of Meinongian objects of thought are relevant for us:

1. The *objectum*, or object of "simple" thoughts: Santa Claus is the objectum of John's act of thinking of Santa Claus. Objecta are the meanings of noun phrases.
2. The *objective*, or object of belief, knowledge, etc.: that Santa Claus is thin is the objective of John's

act of believing that Santa Claus is thin. Objectives are like propositions in that they are the meanings of sentences and other sentential structures.

It is important to note that objecta need not exist and that objectives need not be true. [For details, see: (Meinong, 1904), (Findlay, 1963), (Rapaport, 1976), (Rapaport, 1978), (Rapaport, 1981), (Rapaport, 1982), (Castaneda, 1974), (Castaneda, 1975a), (Castaneda, 1975b), (Castaneda, 1975), (Castaneda, 1977), (Castaneda, 1979), (Tomberlin, 1984), and (Routley, 1979); cf. (Rapaport, 1984a), (Parsons, 1980); cf. (Rapaport, 1985b), (Lambert, 1983); cf. (Rapaport, 1986a), (Zalta, 1983).]

This is, perhaps, somewhat arcane terminology for what might seem like AI common sense. But without an underlying theory, such as Meinong's, there is no way to be sure if common sense can be trusted. It is important to note that not only are all represented things intensional, but that they are all objects of CASSIE's mental acts; that is, they are all in CASSIE's mind (her "belief space") - they are all intentional. Thus, even if CASSIE represents the beliefs of someone else (for example, John's belief that Lucy is rich, as in the conversation in the next section), the objects that she represents as being in that person's mind (as being in his "belief space") are actually CASSIE's representations of those objects - that is, they are in CASSIE's mind.

11.3.2. A conversation with CASSIE

Before giving the syntax and semantics of the case frames employed in representing CASSIE's "mind", we present a conversation we had with her. An ATN parser/generator (Shapiro, 1982) was used to parse the English input into SNePS and to generate the SNePS structures in English. User input is on the lines beginning with the :-prompt; CASSIE's output is on the lines that follow. (Execution and garbage-collection times have been edited out for clarity.)

```
: Young Lucy petted a yellow dog
I understand that young Lucy petted a yellow dog

: What is yellow
```

a dog is yellow

: Dogs are animals
I understand that dogs are animals

: Who petted an animal
young Lucy petted a yellow dog

: Lucy is sweet
I understand that young Lucy is sweet

: Lucy is a girl
I understand that sweet young Lucy is a girl

: Girls are people
I understand that girls are people

: What was petted by a person
a yellow dog was petted by sweet young Lucy

: The dog is named Rover
I understand that Rover is a yellow dog

: What was petted by a person
yellow Rover was petted by sweet young Lucy

: John believes that Lucy is rich
I understand that John believes that Lucy is rich

: Who is rich
I don't know.

: Who is sweet
young Lucy is sweet

: John believes that Lucy is old
I understand that John believes that rich Lucy is old

: John is a boy
I understand that John is a boy

: Boys are people
I understand that boys are people

: Dogs are pets
I understand that dogs are pets

: For every p and d if p is a person and d is a pet then p
loves d
I understand that for every d and p, if p is a person
and
d is a pet

then p loves d

: Who loves a pet
 sweet young Lucy loves yellow Rover
 and
 John loves yellow Rover

11.3.3. Syntax and semantics of SNePS/CASSIE

In this section, we give the syntactic formation rules (SR) and semantic interpretations (SI) for the nodes and arcs used in this interaction, together with some other important ones. We return to a more detailed examination of the interaction in the next section. What we present here is our current model; we make no claims about the completeness of the representational scheme. In particular, we leave for another paper a discussion of such structured individuals as the golden mountain or the round square, which raise difficult and important problems with predication and existence. [For a discussion of these issues, see (Rapaport, 1978), (Rapaport, 1985a).]

Information is represented in SNePS by means of *nodes* and *arcs*. Since the meaning of a node is determined by what it is connected to in the network, there are no isolated nodes. Nodes that only have arcs pointing to them are considered to be unstructured or *atomic*. They include:

- (A1) *sensory* nodes, which represent interfaces with the external world (in the examples that follow, they will represent words, sounds, or utterances);
- (A2) *base* nodes, which represent constant individual concepts and properties;
- (A3) *variable* nodes, which represent arbitrary individuals (cf. (Fine, 1983)) or arbitrary propositions.

Molecular nodes, which have arcs emanating from them, include:

- (M1) *structured individual* nodes, which represent structured individual concepts or properties (that is, concepts and properties represented in such a way that their internal structure is exhibited; see the discussion of structured information in (Woods, 1975));
- (M2) *structured proposition* nodes, which represent

propositions; those with no incoming arcs represent *beliefs* of the system.³⁴ (Note that structured proposition nodes can also be considered to be structured individuals.) Proposition nodes are either *atomic* (representing atomic propositions) or are *rule nodes*. Rule nodes represent deduction rules and are used by SNIP (the SNePS Inference Package) for node-based deductive inference.³⁵

For each of the three categories of molecular nodes (structured individuals, atomic propositions, and rules), there are *constant* nodes of that category and *pattern* nodes of that category representing arbitrary entities of that category.

The rules labeled 'SR', below, should be considered as syntactic formation rules for a *non-linear* network language. The semantic interpretations, labeled 'SI', are in terms of Meinongian objecta and objectives, which are intentional objects, that is, objects of thought. Since intentional objects are intensional, our Meinongian semantics is an *extensional* semantics over a domain of *intensional* entities (Meinongian objects).

We begin with a few definitions.³⁶

Definition 1

A node *dominates* another node if there is a path of directed

³⁴There is a need to distinguish structured proposition nodes with no incoming arcs from structured individual nodes with no incoming arcs; the latter, of course, are not beliefs of the system. This is handled by the syntactic formation rules and their semantic interpretations. There is also a need to distinguish between beliefs of the system and those propositions that the system is merely contemplating or "assuming" temporarily [cf. (Meinong, 1983)]. We are currently adding this capability to SNePS by means of an *assertion* operator ('!').

³⁵For details, see (Shapiro, 1977), (Shapiro, 1978), (McKay and Shapiro, 1980), (McCarty and Sridharan, 1981), (Shapiro and McKay, 1980), (Shapiro, Martins, and McKay, 1982), (Martins, 1983a).

³⁶These are actually only rough definitions; the interested reader is referred to (Shapiro, 1979b), Section 2.1, for more precise ones.

arcs from the first node to the second node.

Definition 2

A *pattern* node is a node that dominates a variable node.

Definition 3

An *individual* node is either a base node, a variable node, or a structured constant or pattern individual node.

Definition 4

A *proposition* node is either a structured proposition node or an atomic variable node representing an arbitrary proposition.

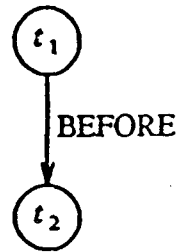
SR.1 If "*w*" is an English word and "*i*" is an identifier not previously used, then



is a network, *w* is a sensory node, and *i* is a structured individual node.

SI.1 *i* is the Meinongian objectum corresponding to the utterance of *w*.

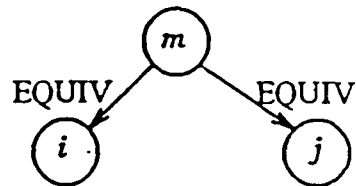
SR.2 If either "*t*₁" and "*t*₂" are identifiers not previously used, or "*t*₁" is an identifier not previously used and *t*₂ is a temporal node, then



is a network and t_1 and t_2 are *temporal* nodes, that is individual nodes representing times.

SI.2 t_1 and t_2 are Meinongian objecta corresponding to two time intervals, the former occurring before the latter.

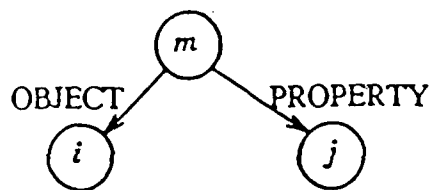
SR.3 If i and j are individual nodes, and " m " is an identifier not previously used, then



is a network and m is a structured proposition node.

SI.3 m is the Meinongian objective corresponding to the proposition that Meinongian objecta i and j (are believed by CASSIE to) correspond to the same actual object. (This is not used in the conversation, but is needed for fully intensional representational systems: cf. (Rapaport, 1978;RAPA84b) and (Castaneda, 1974:CAST75b) for analyses of this sort of relation, and (Maida and Shapiro, 1982) for a discussion of its use.)

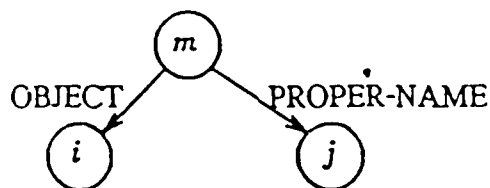
SR.4 If i and j are individual nodes and " m " is an identifier not previously used, then



is a network and m is a structured proposition node.

SI.4 m is the Meinongian objective corresponding to the proposition that i has the property j .

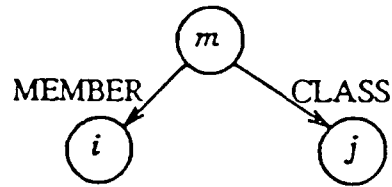
SR.5 If i and j are individual nodes and " m " is an identifier not previously used, then



is a network and m is a structured proposition node.

SI.5 m is the Meinongian objective corresponding to the proposition that Meinongian objectum i 's proper name is j . (j is the Meinongian objectum that is i 's proper name: its expression in English is represented by a node at the head of a LEX-arc emanating from j .)

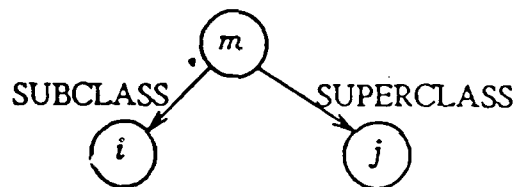
SR.6 If i and j are individual nodes and " m " is an identifier not previously used, then



is a network and m is a structured proposition node.

SI.6 m is the Meinongian objective corresponding to the proposition that i is a (member of class) j .

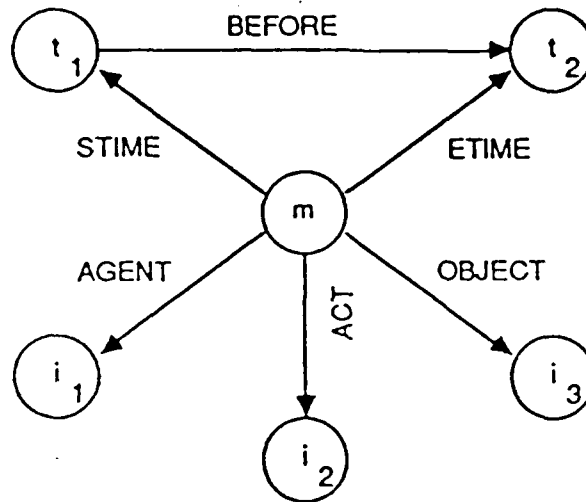
SR.7 If i and j are individual nodes and " m " is an identifier not previously used, then



is a network and m is a structured proposition node.

SI.7 m is the Meinongian objective corresponding to the proposition that (the class of) i is are (a subclass of the class of) j s.

SR.8 If i_1, i_2, i_3 are individual nodes, t_1, t_2 are temporal nodes, and " m " is an identifier not previously used, then

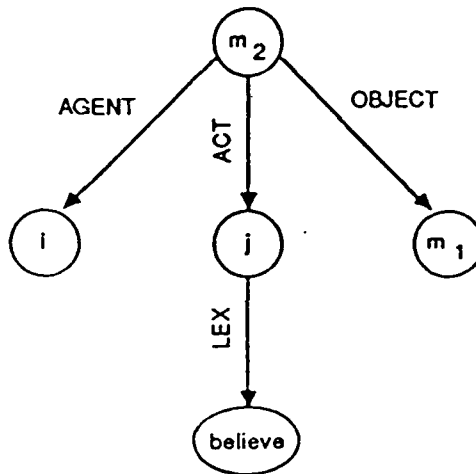


is a network and m is a structured proposition node.

SL.8 m is the Meinongian objective corresponding to the proposition that agent i_1 performs act i_2 to or on i_3 starting at time t_1 and ending at time t_2 , where t_1 is before t_2 .

It should be noted that the ETIME and STIME arcs are optional and can be part of any proposition node. They are a provisional technique for handling the representation of acts and events; our current research on temporal representation is much more complex and is discussed in Section 11.4.7, below.

SR.9 If m_j is a proposition node, i is an individual node, j is the (structured individual) node with a LEX arc to the node, believe, and " m_2 " is an identifier not previously used, then

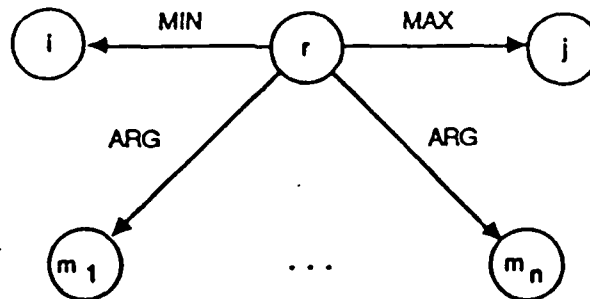


is a network and m_2 is a structured proposition node.

SI.9 m_2 is the Meinongian objective corresponding to the proposition that agent i believes proposition m_1 .

Two special cases of SR.9 that are of interest concern *de re* and *de dicto* beliefs; they are illustrated in Figure 11-2 and Figure 11-3. [For details, see (Rapaport and Shapiro, 1984) and (Rapaport, 1984b), (Rapaport, 1986b).]

SR.10 If m_1, \dots, m_n are proposition nodes ($n \geq 0$), " i " and " j " are integers between 0 and n , inclusive, and " r " is an identifier not previously used, then



is a network, and r is a rule node.

SI.10 r is the Meinongian objective corresponding to the proposition that there is a relevant connection between

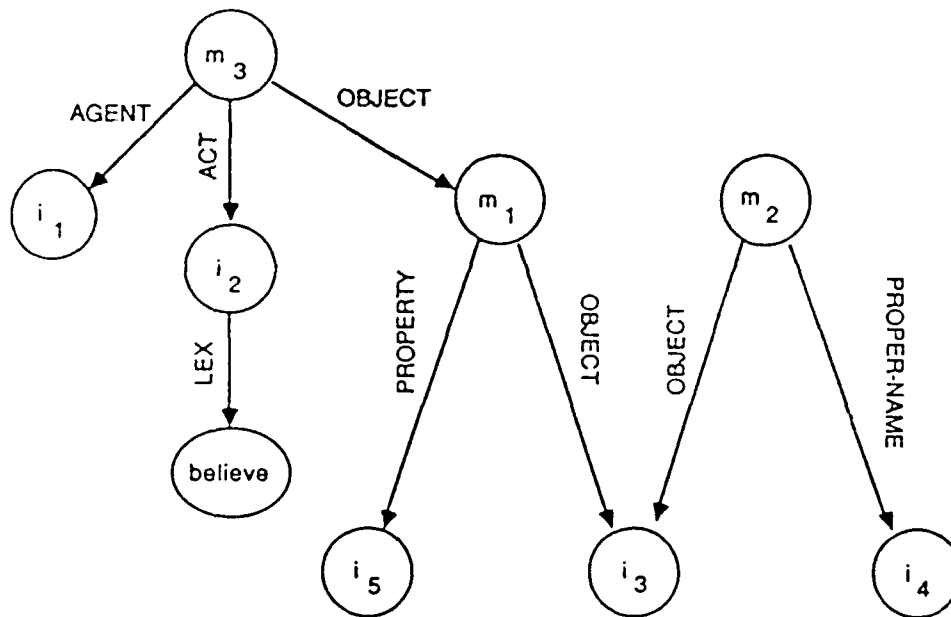


Figure 11-2: Meinongian Objective - *de re* Reading

- m_3 is the Meinongian objective corresponding to the proposition that agent i_1 believes *de re* of objectum i_3 (who is believed by CASSIE to be named i_4) that it has the property i_5 .

propositions m_1, \dots, m_n such that at least i and at most $i(j)$ of them are simultaneously true.

Rule r of SR/SI.10 is called *AND-OR* and is a unified generalization of negation ($i = j = 0$), binary conjunction ($i = j = 2$), binary inclusive disjunction ($i = 1, j = 2$), binary exclusive disjunction ($i = 0, j = 1$), etc.

SR.11 If m_1, \dots, m_n are proposition nodes ($n \leq 0$), is an integer between 0 and n , inclusive, and " r " is an identifier not previously used, then

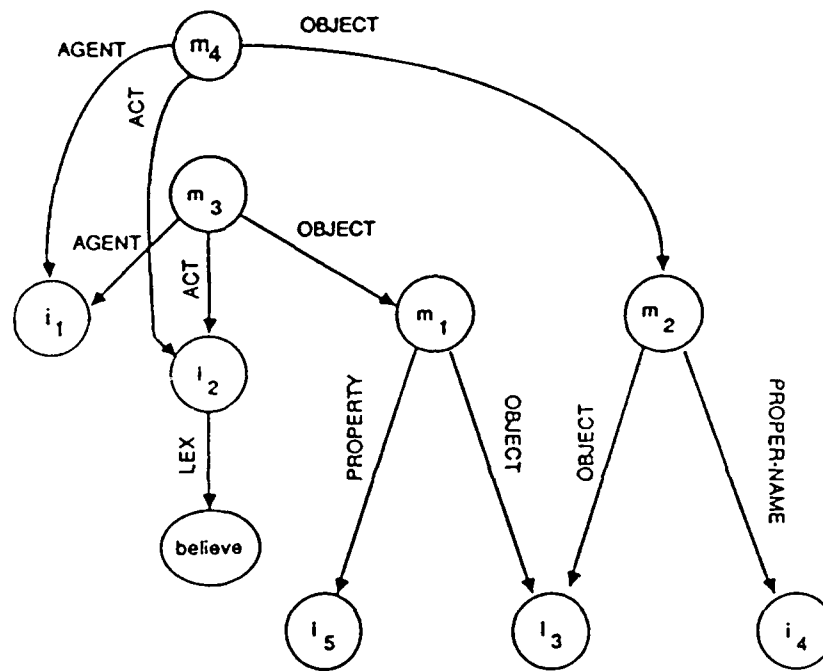
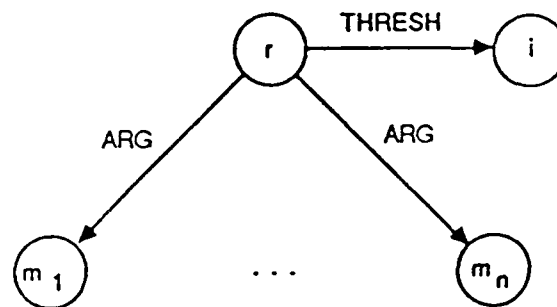


Figure 11-3: Meinongian Objective - *de dicto* Reading

m_4 is the Meinongian objective corresponding to the proposition that agent i_1 believes *de dicto* that objectum i_3 (who is believed by i_1 to be named i_4) has the property i_5 .

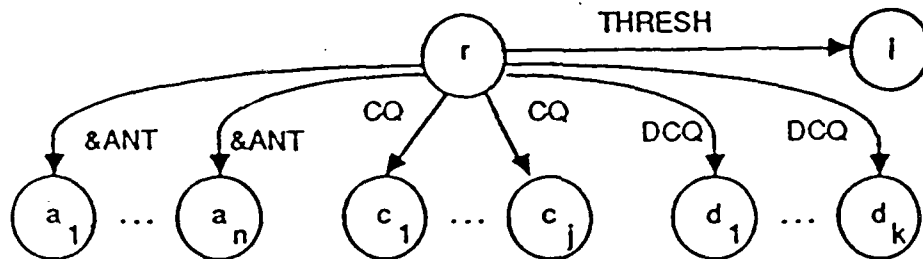


is a network, and r is a rule node.

SI.11 r is the Meinongian objective corresponding to the proposition that there is a relevant connection between propositions m_1, \dots, m_n such that either fewer than i of them are true or they all are true.

Rule r of SR/SI.11 is called *THRESH* and is a generalization of the material biconditional ($i = 1$).

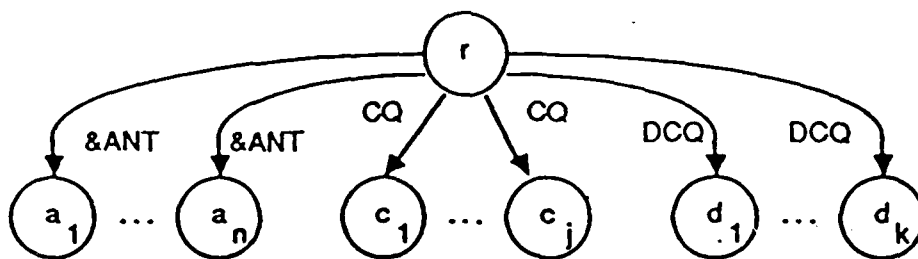
SR.12 If $a_1, \dots, a_n, c_1, \dots, c_j$, and d_1, \dots, d_k are proposition nodes ($n \geq 1; j, k \geq 0; j + k \geq 1$). " i " is an integer between 1 and n , inclusive, and " r " is an identifier not previously used, then



is a network, and r is a rule node.

SI.12 r is the Meinongian objective corresponding to the proposition that the conjunction of any i of the propositions a_1, \dots, a_n relevantly implies each c_l ($1 \leq l \leq j$) and relevantly implies each d_l ($1 \leq l \leq k$) for which there is not a better reason to believe it is false.

SR.13 If $a_1, \dots, a_n, c_1, \dots, c_j$, and d_1, \dots, d_k are proposition nodes ($n, j, k \geq 0$), and " r " is an identifier not previously used, then



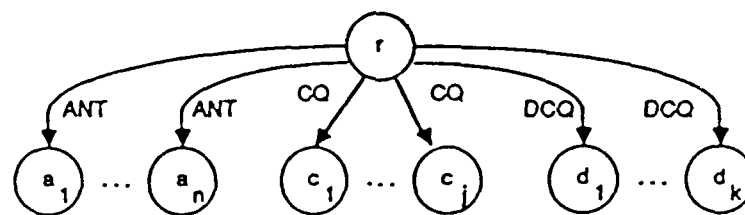
is a network, and r is a rule node.

SI.13 r is the Meinongian objective corresponding to the proposition that the conjunction of the propositions a_1, \dots, a_n relevantly implies each c_l ($1 \leq l \leq j$) and relevantly implies

each d_l ($1 \leq l \leq k$) for which there is not a better reason to believe it is false.

The d_l are *default* consequences. in the sense that each is implied only if it is neither the case that CASSIE already believes *not* d_l nor that *not* d_l follows from non-default rules.

SR.14 If $a_1, \dots, a_n, c_1, \dots, c_j$, and d_1, \dots, d_k are proposition nodes ($n \geq 1; j, k \geq 0; j + k \geq 1$), and " r " is an identifier not previously used, then



is a network, and r is a rule node.

SL.14 r is the Meinongian objective corresponding to the proposition that any a_i , $1 \leq i \leq n$, relevantly implies each c_l ($1 \leq l \leq j$) and relevantly implies each d_l ($1 \leq l \leq k$) for which there is not a better reason to believe it is false.

SR.15 If m is a proposition node, and " r " is an identifier not previously used, then



is a network, and r is a rule node.

SI.15 r is the Meinongian objective corresponding to the proposition that there is no good reason for believing proposition m .

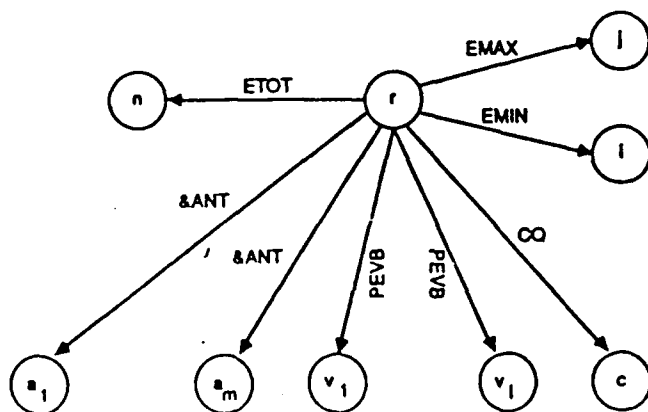
SR.16 If r is a rule node as specified by SR.10-SR.15, and r dominates variable nodes v_1, \dots, v_n , and, in addition, arcs labeled "AVB" go from r to each v_i , then r is a quantified rule node.

SI.16 r is the Meinongian objective corresponding to the proposition that the rule that would be expressed by r without the AVB arcs holds after replacing each v_i by any Meinongian object in its range.

SR.17 If r is a rule node as specified by SR.10-SR.15, and r dominates variable nodes v_1, \dots, v_n , and, in addition, arcs labeled "EVB" go from r to each v_i , then r is a quantified rule node.

SI.17 r is the Meinongian objective corresponding to the proposition that the rule that would be expressed by r without the EVB arcs holds after replacing each v_i by some Meinongian object in its range.

SR.18 If a_1, \dots, a_m and c are proposition nodes; v_1, \dots, v_l are variable nodes dominated by one or more of a_1, \dots, a_m ; c ; "i", "j", and "n" are integers ($0 \leq i \leq j \leq mn$); and " r " is an identifier not previously used; then



is a network, and r is a rule node.

SL18 r is the Meinongian objective corresponding to the proposition that, of the n sequences of Meinongian objects which, when substituted for the sequence v_1, \dots, v_l , make all the a_i believed propositions, between i and j of them also satisfy c . (For further details on such numerical quantifiers, see (Shapiro, 1979c).)

11.3.4. The conversation with CASSIE, revisited

In this section, we shall review the conversation we had with CASSIE, showing the network structure as it is built - that is, showing the structure of CASSIE's mind as she is given information and as she infers new information. (Comments are preceded by a dash.)

: Young Lucy petted a yellow dog

I understand that young Lucy petted a yellow dog

- CASSIE is told something, which she now believes. Her entire belief structure is shown in Figure 11-4 (a). The node labeled "now" represents the current time, so the petting is clearly represented as being in the past. CASSIE's response is "I understand that" appended to her English description of the proposition just entered.

: What is yellow

a dog is yellow

- This response shows that CASSIE actually has some beliefs; she did not just parrot back the above sentence.

: Dogs are animals

I understand that dogs are animals

- CASSIE is told a small section of a class hierarchy.

: Who petted an animal

young Lucy petted a yellow dog

- CASSIE can answer the question using the class hierarchy, because, prior to the conversation, the inheritance rule

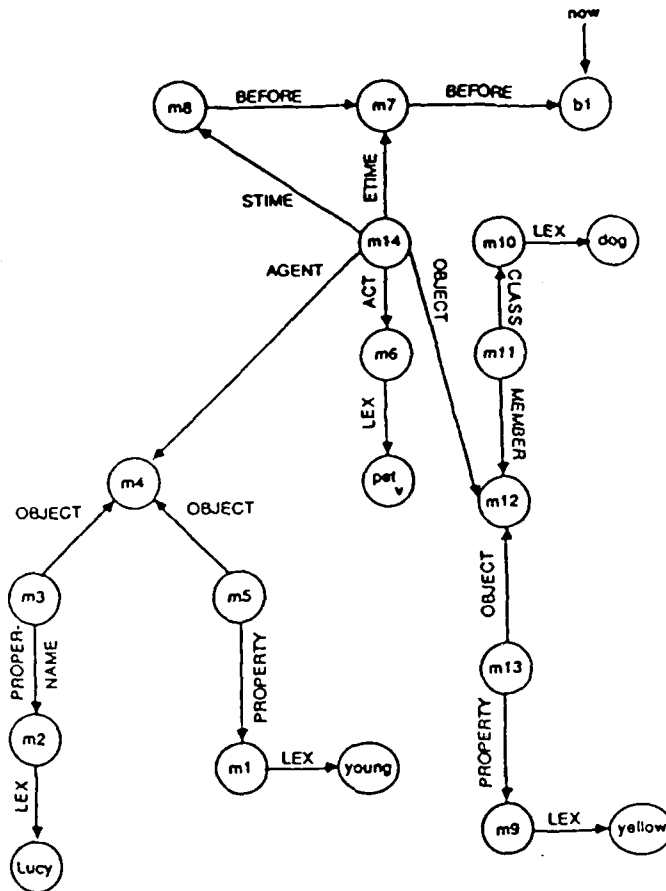


Figure 11-4: Fragment of CASSIE's Belief Structure

Fragment of CASSIE's belief structure after being told that young Lucy petted a yellow dog.

```

(def-path class (compose class (kstar
  (compose subclass- superclass))))
  
```

was given to SNePS. This rule says that the CLASS arc is implied by the path consisting of a CLASS arc followed by zero or more occurrences of the two-arc path consisting of the converse SUBCLASS arc followed by the SUPERCLASS arc [see (Shapiro, 1978), (Srihari, 1981)]. The dog was called "a yellow dog" rather than "a yellow animal" because the redundant CLASS arc is not built. Figure 11-5 shows the current state of CASSIE's belief structure about the dog's classification and color.

: Lucy is sweet
 I understand that young Lucy is sweet

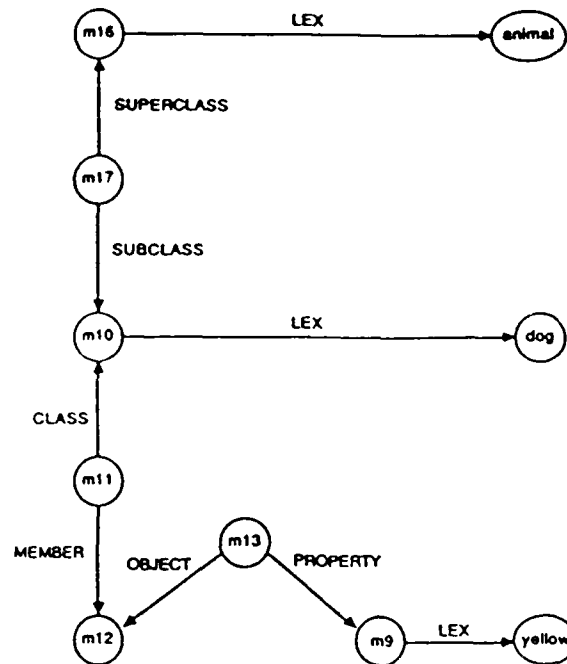


Figure 11-5: CASSIE's Belief Structure

CASSIE's belief structure about the dog's classification and color. (Node m12 represents the dog.)

- CASSIE's response shows that she identifies this Lucy with the previous Lucy.

: Lucy is a girl
I understand that sweet young Lucy is a girl

- The beginning of a class hierarchy for Lucy. Notice that all the adjectival properties of Lucy are mentioned.

: Girls are people
I understand that girls are people

- More of the class hierarchy is given.

: What was petted by a person
a yellow dog was petted by sweet young Lucy

- Again, the proposition is retrieved using the CLASS inheritance rule. The answer is expressed in the passive voice because of the way the question was asked.

: The dog is named Rover
I understand that Rover is a yellow dog

- 'the dog' refers to the only dog CASSIE knows about, who is now given a name.

: What was petted by a person
yellow Rover was petted by sweet young Lucy

- This is exactly the same question that was asked before. It is answered differently this time, because the dog now has a name, and CASSIE prefers to describe an individual by its name when it has one.

: John believes that Lucy is rich
I understand that John believes that Lucy is rich

- At this point in our development of CASSIE, she interprets 'believes that' contexts to be *de dicto*, so she assumes that the Lucy that John has in mind is a different one from the Lucy that she knows. Figure 11-6 shows CASSIE's beliefs about the two Lucies.

: Who is rich
I don't know.

- CASSIE knows no one who is rich. She only believes that *John* believes that someone (whom she believes that he believes to be named 'Lucy') is rich. The answer is 'I don't know', rather than 'no one is rich', because CASSIE doesn't use the closed-world hypothesis.

: Who is sweet
young Lucy is sweet

- This question is asked merely to demonstrate that Lucy is able to answer a "who is <property>" question when she has relevant beliefs.

: John believes that Lucy is old
I understand that John believes that rich Lucy is old

- Even though CASSIE assumes that John knows a different Lucy than she knows, she assumes that all John's beliefs about "Lucy" are about the same Lucy.

: John is a boy
I understand that John is a boy

- This and the next two inputs are given to establish more of the class hierarchy and

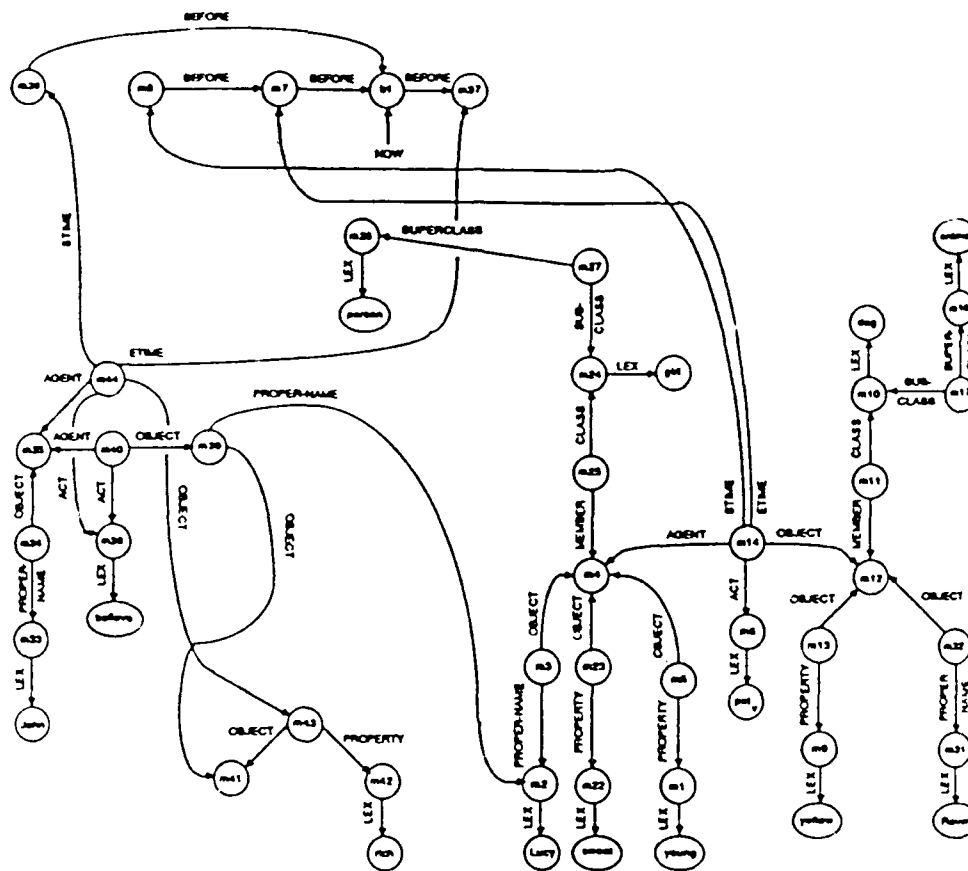


Figure 11-6: A Fragment of the Network

A Fragment of the network after CASSIE is told that John believes that Lucy is rich, showing CASSIE's beliefs about the two Lucies.

to make it clear that when CASSIE answers the last question of this session, she is doing both path-based reasoning and node-based reasoning at the same time.

I understand that boys are people

: Dogs are pets

I understand that dogs are pets

: For every p and d if p is a person and d is a pet then p loves d

I understand that for every d and p, if p is a person and
d is a pet

- Figure 11-7 shows how this node-based rule fits into the class hierarchy. This is, we believe, equivalent to the integrated TBox/ABox mechanism proposed for KRYPTON [(Brachman, Fikes, and Levesque, 1983), (Brachman, Gilbert, and Levesque, 1985)].

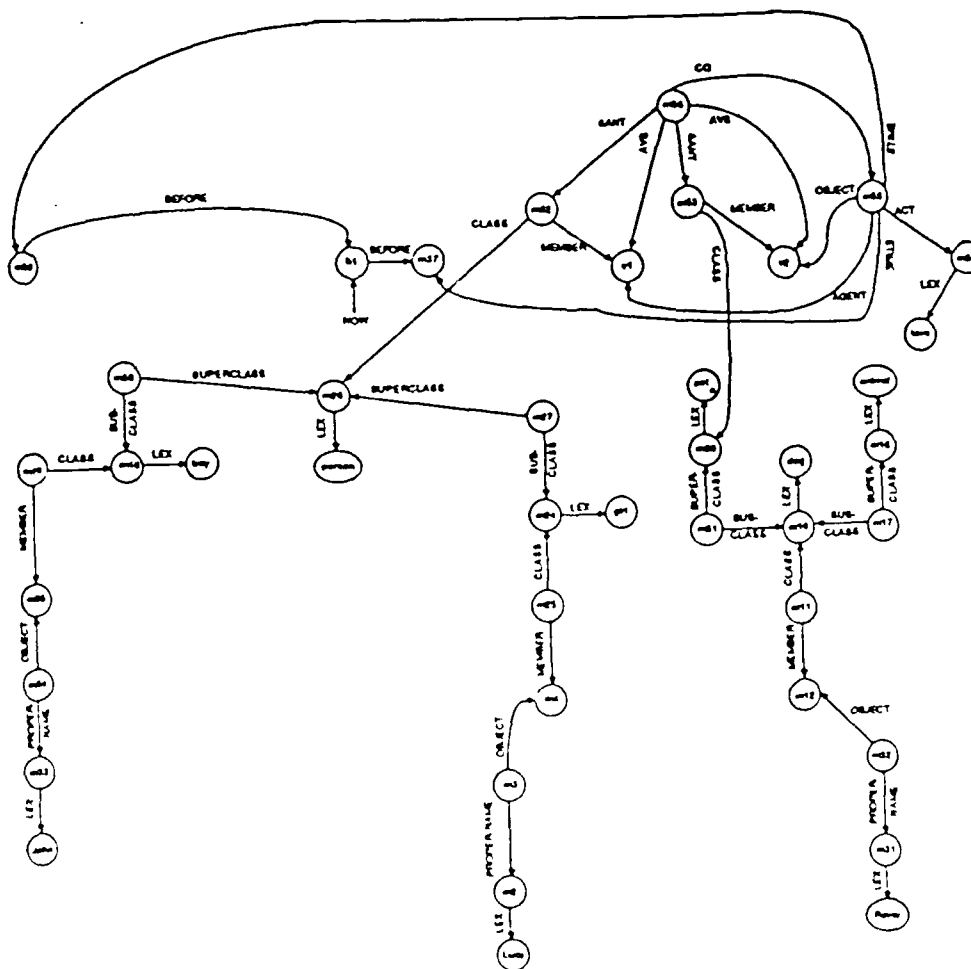


Figure 11-7: A Node-based Rule in a Class Hierarchy

: Who loves a pet
sweet young Lucy loves yellow Rover
and

John loves yellow Rover

- The question was answered using path-based inferencing to deduce that Lucy and John are people and that Rover is a pet, and node-based inferencing to conclude that, therefore, Lucy and John love Rover.
- The full network showing CASSIE's state of mind at the end of the conversation is given in Figure 11-8.

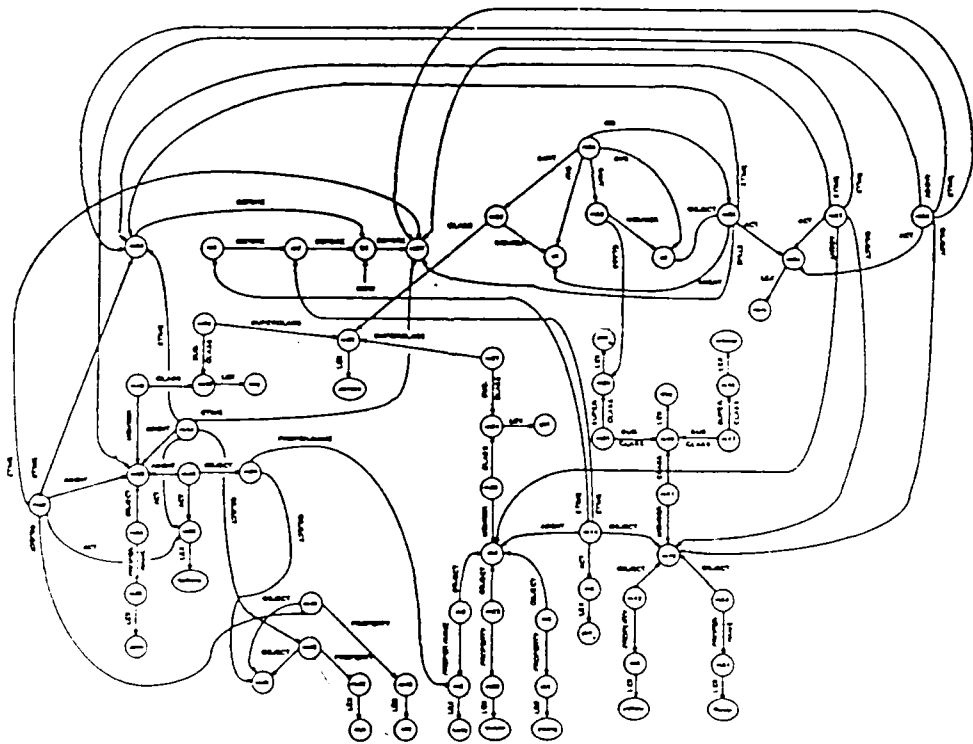


Figure 11-8: CASSIE's Beliefs at the End of the Conversation

11.4. Extensions and applications of SNePS

In this essay, we have been advocating the use and interpretation of SNePS networks to model (the beliefs of) a cognitive agent. SNePS, however, is of much wider and more general applicability. In this section, we give examples of recent and current research projects using SNePS in belief-revision, as a database management system, for developing several expert systems, and for representing temporal information in narratives. Even though most of these uses of SNePS do not explicitly involve a cognitive agent, it should be noted that in each case the asserted nodes can be treated as "beliefs" of the system: beliefs about the database, beliefs about the various domains of the expert systems, beliefs about linguistics, etc.

11.4.1. SNePS as a database management system

SNePS can be used as a network version of a relational database in which every element of the relational database is represented by an atomic node, each row of each relation is represented by a molecular node, and each column label (attribute) is represented by an arc label. Whenever a row r has an element e in column c , the molecular node representing r has an arc labeled c pointing to the atomic node representing e . Relations (tables) may be distinguished by either of two techniques, depending on the particular relations and attributes in the relational database. If each relation has an attribute that does not occur in any other relation, then the presence of an arc labeled with that attribute determines the relationship represented by the molecular node. A review of the syntax of the CASSIE networks will show that this technique is used there. The other technique is to give every molecular node an additional arc (perhaps labeled "RELATION") pointing to an atomic node whose identifier is the name of the relation. Table 11-1 shows the Supplier-Part-Project database of (Date, 1981, p 114). Notice that the SNAME and STATUS attributes only occur in the SUPPLIER relation; PNAME, COLOR, and WEIGHT only occur in the PART relation; JNAME only occurs

in the PROJECT relation; and QTY only occurs in the SPJ relation. Figure 11-9 shows the SNePS network for part of this database.

Table 1: SUPPLIER

S#	SNAME	STATUS	CITY
s1	Smith	20	London
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	London
s5	Adams	30	Athens

Table 2: PART

P#	PNAME	COLOR	WEIGHT	CITY
p1	nut	red	12	London
p2	bolt	green	17	Paris
p3	screw	blue	17	Rome
p4	screw	red	14	London
p5	cam	blue	12	Paris
p6	cog	red	19	London

Table 3: PROJECT

J#	JNAME	CITY
j1	sorter	Paris
j2	punch	Rome
j3	reader	Athens
j4	console	Athens
j5	collator	London
j6	terminal	Oslo
j7	tape	London

Table 4: SPJ

S#	P#	J#	QTY
s1	p1	j1	200
s1	p1	j4	700
s2	p3	j1	400
s2	p3	j2	200
s2	p3	j3	200
s2	p3	j4	500
s2	p3	j5	600
s2	p3	j6	400
s2	p3	j7	800
s2	p5	j2	100
s3	p3	j1	200
s3	p4	j2	500
s4	p6	j3	300
s4	p6	j7	300
s5	p2	j2	200
s5	p2	j4	100
s5	p5	j5	500
s5	p5	j7	100
s5	p6	j2	200
s5	p1	j4	1000
s5	p3	j4	1200
s5	p4	j4	800
s5	p5	j4	400
s5	p6	j4	500

Table 11-1: Tables Supplier Part Project and SPJ

Many database retrieval requests may be formulated using the find command of SNePSUL, the SNePS User's Language. The syntax of find is (find r_1 n_1 ... r_m n_m), where r_i is either an arc or a path, and n_i is either a node or a set of nodes (possibly the value of a nested call to find). The value of a call to find is the set of all nodes in the network with an r_1 arc to any node in the set n_1 , an r_2 arc to any node in

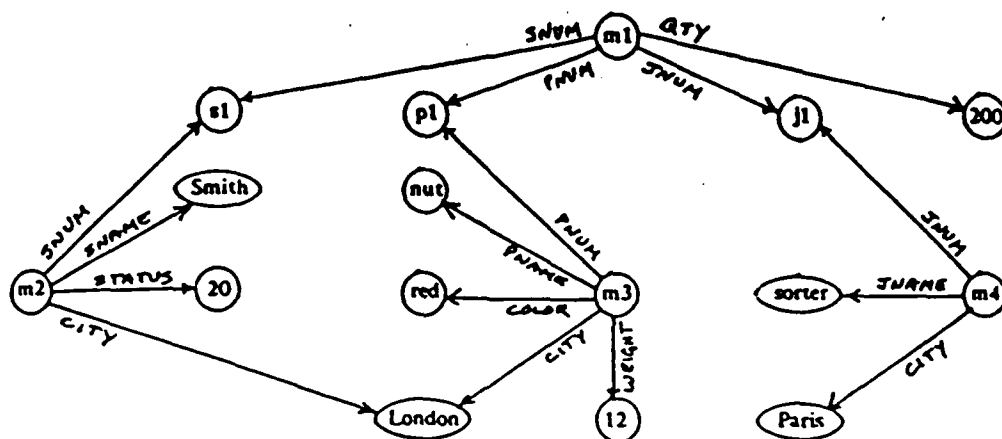


Figure 11-9: Fragment of SNePS Network for the Supplier-Part-Project Database.

the set n_2, \dots , and an r_m arc to any node in the set n_m . Free variables are prefixed by "?". An infix '-' sign between finds represents the set difference operator.

The session below shows some of the queries from (Date, 1981): pp 141-142 translated into find commands, and the results on the database shown above. (In each interaction, comments are preceded by semicolons, user input follows the '*'-prompt, and SNePS responses are on succeeding lines. Execution and garbage collection times have been edited out for clarity.)

```
; Get full details of all projects in London.
* (dump (find jname ?x city London))
(m18 (city (London)) (jname (tope)) (jnum (j7)))
(m16 (city (London)) (jname (collator)) (jnum (j5)))
(dumped)

; Get SNUM values for suppliers who supply project J1
; with part P1.
* (find snum- (find jnum j1 pnum p1))
(s1)

; Get JNAME values for projects supplied by supplier S1.
```

```

• (find (jname- jnum jnum- snum) s1)
(console sorter)

; Get S# values for suppliers who supply both projects
; J1 and J2.
• (find (snum- jnum) j1 (snum- jnum) j2)
(s3 s2)

; Get the names of the suppliers who supply project J1
; with a red part.
• (find (sname- snum snum-) (find jnum j1 (pnum pnum-
color) red))
(Smith)

; Get S# values for suppliers who supply a London or Paris
; project with a red part.
• (find snum- (find (jnum jnum- city) (London Paris)
(pnum pnum- color) red))
(s4 s1)

; Get P# values for parts supplied to any project by
; a supplier in the same city.
• (find pnum- (find (jnum jnum- city) ?city (snum snum-
city) ?city))
(p5 p4 p1 p2 p6 p3)

; Get J# values for projects not supplied with any red part
; by any London supplier.
• ((find jnum- ?x)-(find jnum- (find (pnum pnum- color) red
(snum snum- city) London)))
(j6 j5 j2)

; Get S# values for suppliers supplying at least one part
; supplied by at least one supplier who supplies at least
; one red part.
• (find (snum- pnum pnum- snum snum- pnum pnum- color) red)
(s3 s4 s2 s5 s1)

; Get J# values for projects which use only parts which are
; available from supplier S1.
• ((find jnum- (find qty ?q))
- (find (jnum- pnum) (find pnum- ?r) - (find (pnum- snum)
s1)))
nil

```

11.4.2. Address recognition for mail sorting

A research group led by Sargur N. Srihari is studying address recognition techniques for automated mail sorting (Srihari, Sargur, Jonathan, Palumbo, Niyogi, and Wang, 1985).

Computer determination of the sort-destination of an arbitrary piece of letter-mail from its visual image is a problem that remains far from solved. It involves overcoming several sources of ambiguity at both the spatio-visual and linguistic levels: The location of the destination address has to be determined in the presence of other text and graphics; relevant address lines have to be isolated when there are irrelevant lines of text in the address block; the iconic shapes of characters have to be classified into words of text when numerous types of fonts, sizes, and printing media are present; and the recognized words have to be verified as having the syntax and semantics of an address.

Spatial relationships between objects are essential knowledge sources for vision systems. This source extends naturally to the postal-image understanding problem, because of strong directional expectations. For example, the postage mark is usually above and to the right of the destination address, and the return address is usually to the left of the postage. A semantic network is a natural representation for geometric relations.

An envelope image is segmented into blocks, and a SNePS network is built that represents the geometric relations between blocks and information about the relative and absolute area occupied by each block. A preliminary set of geometric relations are the eight compass points. Relative area occupancy is expressed as the percentage of each block that falls in each of nine equal rectangular subdivisions of the envelope image, and absolute area is given in terms of the number of pixels covered by each block. The program constructs an exhaustive representation of all the geometric relations present in the image. Given the image produced by an initial segmentation procedure, a rough, intuitive output, shown in Figure 11-10 with some arc labels removed for clarity) was produced.

Future work in this area includes refinement of the data structure to represent more information more efficiently and the addition of inferencing capabilities whose objective is to present the control structure with tentative decisions about the address block based only on the information provided by the initial segmentation.

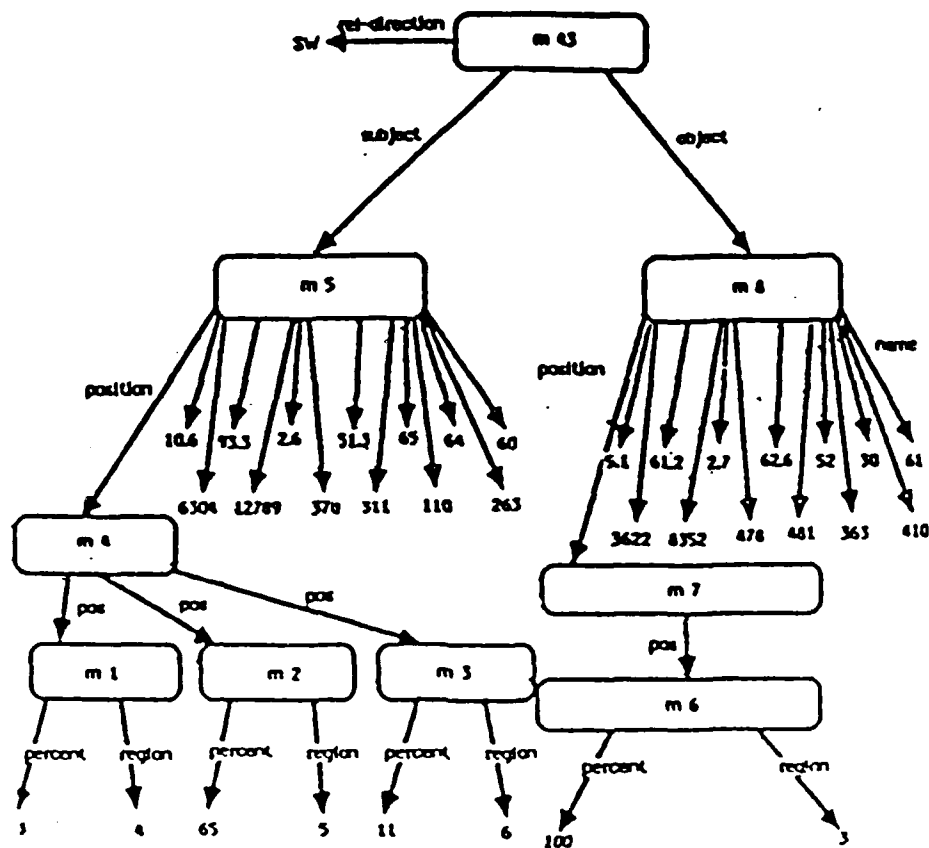


Figure 11-10: SNePS Network Representation of Initial Segmentation of Envelope Image (from Srihari, Hull et al. 1985)

11.4.3. NEUREX

The NEUREX project (Cheng, 1984). (Xiang and Srihari, 1985). (Xiang, Srihari, Shapiro and Chutkow, 1984). (Suchin, 1985) is a diagnostic expert system for diseases of the central and peripheral nervous systems; it also deals with information about neuroaffectors, neuroreceptors, and body parts. SNePS is used to represent spatial structures and functions propositionally. Entities are represented topologically by means of proposition nodes expressing an entity's shape, position, etc., and spatial relations are represented by proposition nodes

expressing adjacency, connectivity, direction, etc. This approach integrates structural and functional neuroanatomical information. Moreover, the representation is both propositional and analog. For the peripheral nervous system, there are nodes representing such propositions as that, for example, a sequence of nerve segments are linked at junctions, and that the whole sequence forms a (peripheral) nerve; the network that is built is itself an analog representation of this nerve (and ultimately, together with its neighbors, of the entire peripheral nervous system). See Chapter 15 for further discussion of analog representations. For the central nervous system, there are coordinates in the network representation that can be used to support reasoning by geometrical computation or graphical interfaces.

As one example, the network of Figure 11-11 can be used by the system to determine which muscles are involved in shoulder-joint flexion, using the SNePS User Language request

```
(find (ms- cn) (find jt shoulder-joint mv flexion)).
```

which returns the following list of four nodes:

```
(deltoid pectoralis_major_clavicular_head  
coracobrachialis biceps_brachii)
```

Furthermore, rules, like that shown in Figure 11-12, can be employed and can even include probabilistic information. (Note that node *r* in Figure 11-12 is the SNePS implementation of the IF-THEN rule; cf. (SR.13).)

11.4.4. Representing visual knowledge

The goal of the Versatile Maintenance Expert System (VMES) project is to develop an expert maintenance system that can reason about digital circuits represented graphically (cf. (Shapiro, Srihari, Geller, and Taie, 1986:SSTG86)). A similar perspective on the need for visual knowledge representation is taken by Tsotsos and Shibahara (Chapter 10) and Havens and Mackworth (Chapter 16). The representation is not pixel-oriented; this is a project in visual knowledge representation integrated with more traditional conceptual and propositional knowledge representation. The graphical form of

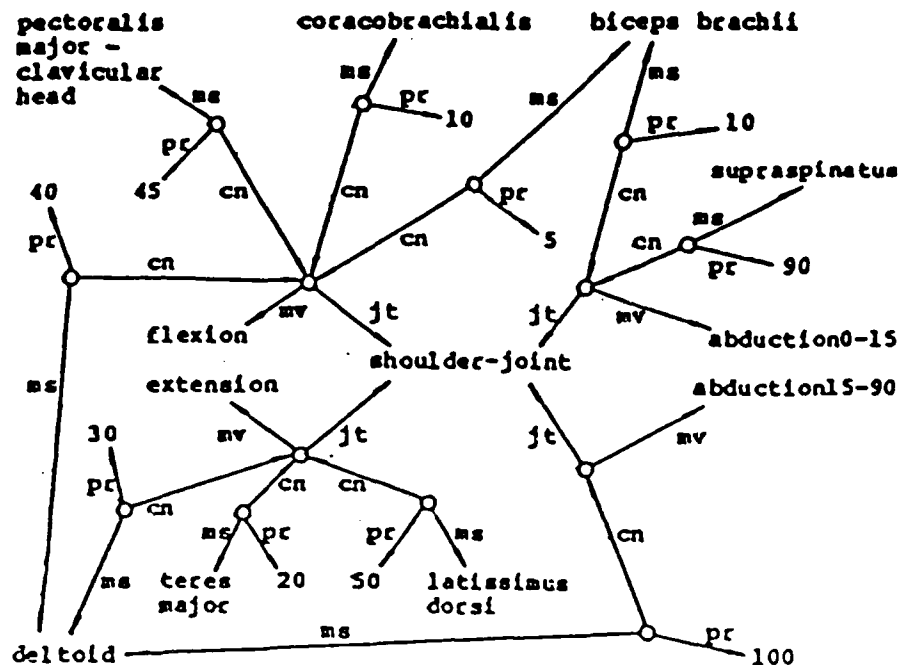


Figure 11-11: Four of the Shoulder-Joint Movements

Four of the shoulder-joint movements with muscles involved and their contribution to each relevant movement. (Meaning of the arc labels: jt=joint; mv=movement; ms=muscle; cn=contribute; pr=percentage.) (From Xiang and Srihari 1985)

an object is a LISP function that, when evaluated, draws the object on the screen. Propositional nodes express information about (1) the relative or absolute position of the object and (2) attributes of the object. Visual knowledge can also be distributed among nodes in traditional hierarchies: for example, the knowledge of how to display a particular hammer may be stored at the level of the class of hammers; the knowledge of how to display a person may be distributed among the nodes for heads, arms, etc.

For example, Figure 11-13 shows a set of three assertions. Node m233 represents the assertion that the object TRIANGLE-1 is 100 units to the right and 20 units below the object SQUARE-1. The MODALITY arc permits the selection of different modes of display; here, we want to display TRIANGLE-1 in "functional" mode. Node m220 states that every member of the class TRIANGLE displayed in functional mode has the form DTRIANG associated with it. Finally, node m219 asserts that TRIANGLE-1 is a TRIANGLE.

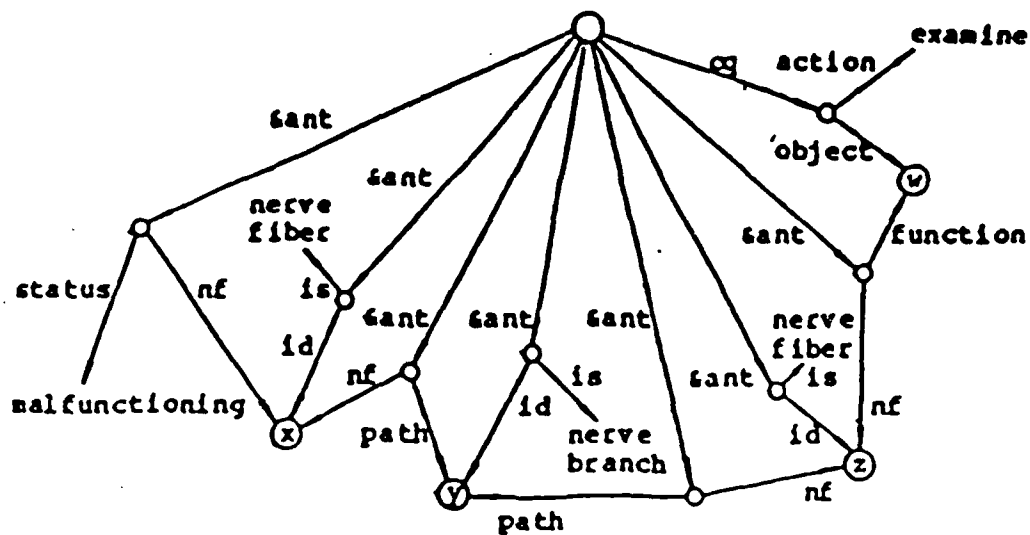


Figure 11-12: SNePS Network for a NEUREX Rule.
(From Xiang and Srihari 1985)

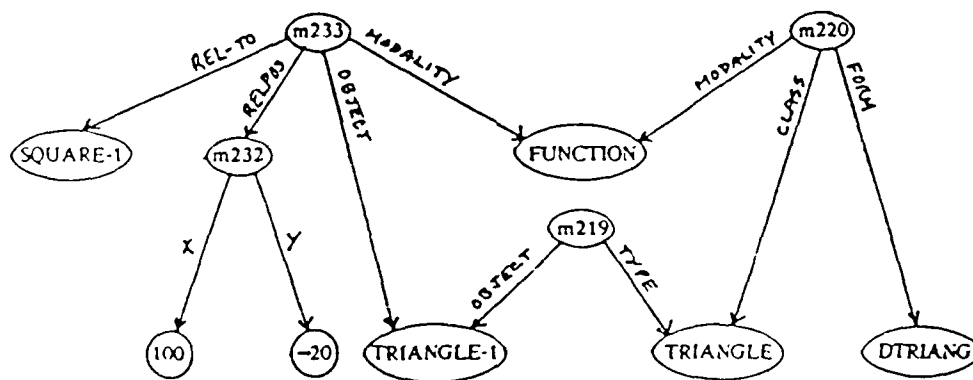


Figure 11-13: SNePS Network in VMES for the Form
and Relative Position of TRIANGLE-1.

Figure 11-14 contains four assertions, of which node m246 is the most complex. It links the object GATE-1 to an absolute position at 100/400 and to the class of all AND-gates. Node m244 asserts that GATE-1 is a part of BOARD-1. Node

m248 asserts that INP1-GATE1 is a PART-OF GATE-1 and belongs to the class AINP1. The label 'PART' actually stands for "has part". Node m239 links the attribute BAD to GATE-1. Every attribute belongs to an attribute class, and the arc ATTRIBUTE-CLASS points to the class STATE.

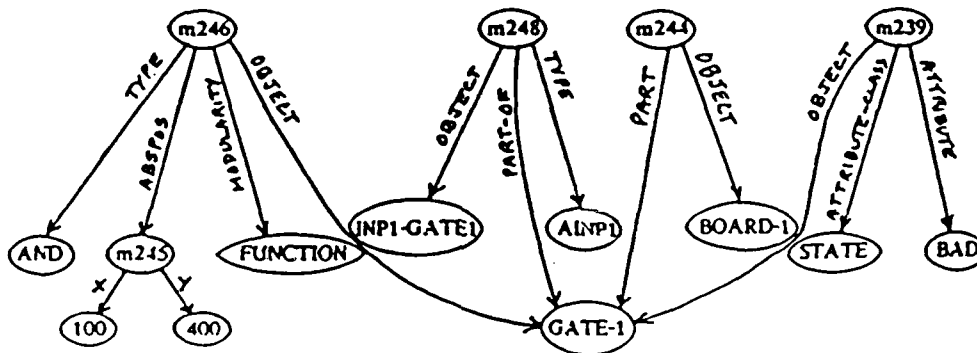


Figure 11-14: SNePS Network in VMES for the Location, Structure, and State of GATE-1.

11.4.5. SNeBR: A belief revision package

The SNePS Inference Package has been extended by João Martins to handle belief revision - an area of AI research concerned with the issues of revising sets of beliefs when a contradiction is found in a reasoning system. Research topics in belief revision include the study of the representation of beliefs, in particular how to represent the notion of belief dependence; the development of methods for selecting the subset of beliefs responsible for contradictions; and the development of techniques to remove some subset of beliefs from the original set of beliefs. (For an overview of the field, see (Martins, 1987).)

SNeBR (*SNePS Belief Revision*) is an implementation in SNePS of an abstract belief revision system called the Multiple Belief Reasoner (MBR), which, in turn, is based on a relevance logic system called SWM (after Shapiro, Wand, and Martins)

(Shapiro and Wand, 1976). (Martins, 1983b). (Martins, 1983a). (Martins and Shapiro, 1984). (Martins and Shapiro, 1986a). (Martins and Shapiro, 1986b). (Martins and Shapiro, 1986c). SWM contains the rules of inference of MBR and defines how contradictions are handled. The only aspect of SWM relevant to this description concerns the objects with which MBR deals, called *supported wffs*. They are of the form

$$A \mid t, o, r$$

where A is a well-formed formula representing a proposition, t is an *origin tag* indicating how A was obtained (for example, as a hypothesis or as a derived proposition), o is an *origin set* containing *all* and *only* the hypotheses used to derive A , and r is a *restriction set* containing information about contradictions *known* to involve the hypotheses in o . The triple t, o, r is called the *support* of the wff A . The origin tag, origin set, and restriction set of a wff are computed when the wff is derived, and its restriction set may be updated when contradictions are discovered.

MBR uses the concepts of context and belief space. A *context* is any set of hypotheses. A context determines a *belief space*, which is the set of all the hypotheses defining the context together with all propositions derived exclusively from them. The propositions in the belief space defined by a given context are characterized by having an origin set that is contained in the context. At any point, the set of all hypotheses under consideration is called the *current context*, which defines the *current belief space*. The only propositions that are retrievable at a given time are the ones belonging to the current belief space.

A contradiction may be detected either because an assertion is derived that is the negation of an assertion already in the network, or because believed assertions invalidate a rule being used (particularly an AND-OR or a THRESH rule; see (SR/SI.10-11)). In the former case, the contradiction is noted when the new, contradictory, assertion is about to be built into the network, since the Uniqueness Principle guarantees that the contradictory assertions will share network structure. In the latter case, the contradiction is noted in the course of applying the rule. In the former case, it may be that the contradictory

assertions are in different belief spaces (only the new one being in the current belief space). If so, the restriction sets are updated to reflect the contradictory sets of hypotheses, and nothing else happens. If the contradictory assertions are both in the current belief space (which will be the case when one of them is a rule being used), then, besides updating the restriction sets, the user will be asked to delete at least one of the hypotheses underlying the contradiction from the current context. Management of origin sets according to SWM guarantees that, as long as the current context was originally not known to be contradictory, removal of any one of the hypotheses in the union of the origin sets of the contradictory assertions from the current context will restore the current context to the state of not being known to be inconsistent.

11.5. Knowledge-based natural language understanding

Jeannette Neal has developed an AI system that can treat knowledge of its own language as its discourse domain. (Neal, 1985). The system's linguistic knowledge is represented declaratively in its network knowledge base in such a way that it can be used in the dual role of "program" to analyze language input to the system and "data" to be queried or reasoned about. Since language forms (part of) its domain of discourse, the system is also able to learn from the discourse by being given instruction in the processing and understanding of language. As the system's language knowledge is expanded beyond a primitive kernel language, instructions can be expressed in an increasingly sophisticated subset of the language being taught. Thus, the system's language is used as its own metalanguage.

The kernel language consists of a relatively small collection of predefined terms and rewrite rules for expressing syntax and for expressing the mapping of surface strings to the representation of their interpretations.

The knowledge representations include representations for surface strings and for relations such as: (a) a lexeme being a member of a certain lexical category, (b) bounded string B

being in category C and this phrase structure being represented by concept N, (c) a structure or parsed string expressing a certain concept, and (d) one phrase structure being a constituent of another structure.

In order to talk about both the syntax and semantics of language, the network representations distinguish between a word or string and its interpretation. In one experiment, the statements

- (1) A WOMAN IS A HUMAN
- (2) 'WOMAN' IS SINGULAR

were input to the system. The first makes a claim about women; the second makes a claim about the word 'woman'. Nodes m40 and m50 of Figure 11-15, respectively, represent the propositions expressed by these statements. The concept or class expressed by 'WOMAN' is represented by node b22; the entity represented by node b22 is a participant in the subset-superset proposition expressed by (1). However, in the representation of (2), the word 'WOMAN' itself is the entity having the property SINGULAR.

Additional statements, such as:

- (R) IF THE HEAD-NOUN OF A NOUN-PHRASE X
HAS NUMBER Y, THEN X HAS NUMBER Y.

were input to the system to demonstrate the use of a subset of English as its own metalanguage in building up the system's language ability from its primitive predefined language. Figure 11-16 illustrates the representation of the system's interpretation of rule (R) as well as the representation of certain linguistic relations. Node m87 represents the proposition that some bounded string represented by variable node v4 is in the category HEAD-NOUN, and this phrase structure is represented by variable node v3. Node m88 represents that the phrase structure represented by node v3 is a constituent of v1, which represents a NOUN-PHRASE structure. (In this figure, the AVB arcs have been eliminated for clarity; cf. (SR/SL16).) As soon as any rule such as (R) is parsed and interpreted, it is immediately available for use in subsequent processing. Thus, the system is continuously educable and can use its language as its own metalanguage.

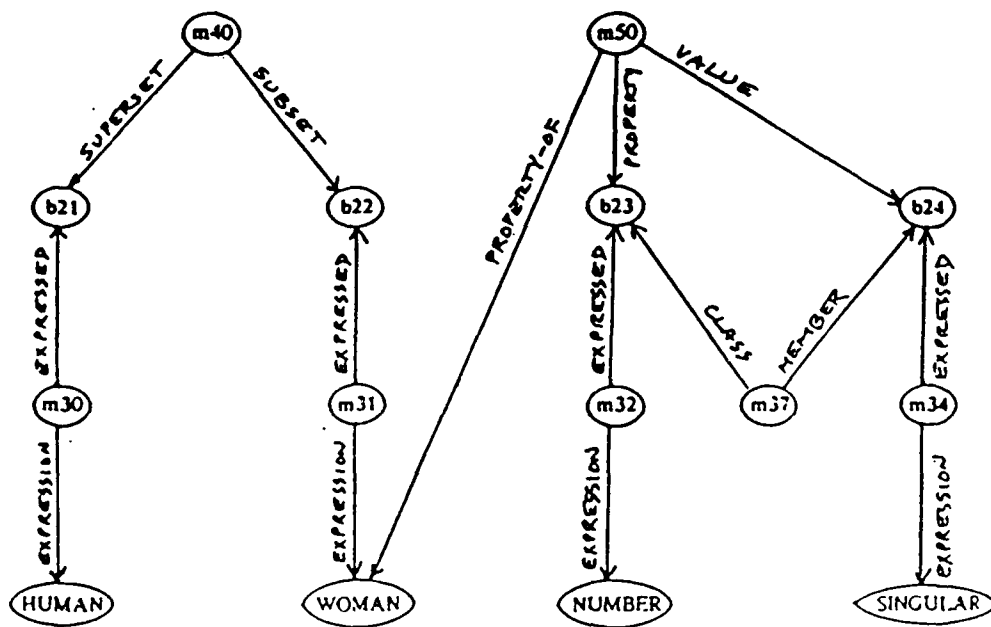


Figure 11-15: Representation of the Interpretation of Statements About Linguistic and Non-linguistic Entities.

11.5.1. Temporal structure of narrative

Michael Almeida is using SNePS in the development of a system that will be able to read a simple narrative text and construct a model of its temporal structure (Almeida and Shapiro, 1983). (Almeida, 1986). This project uses an event-based, rather than a proposition-based, approach: that is, intervals and points of time are associated with events represented as objects in the network rather than with the propositions that describe them. The temporal model itself consists of these intervals and points of time related to one another by such relations as BEFORE/AFTER, DURING/CONTAINS, etc.

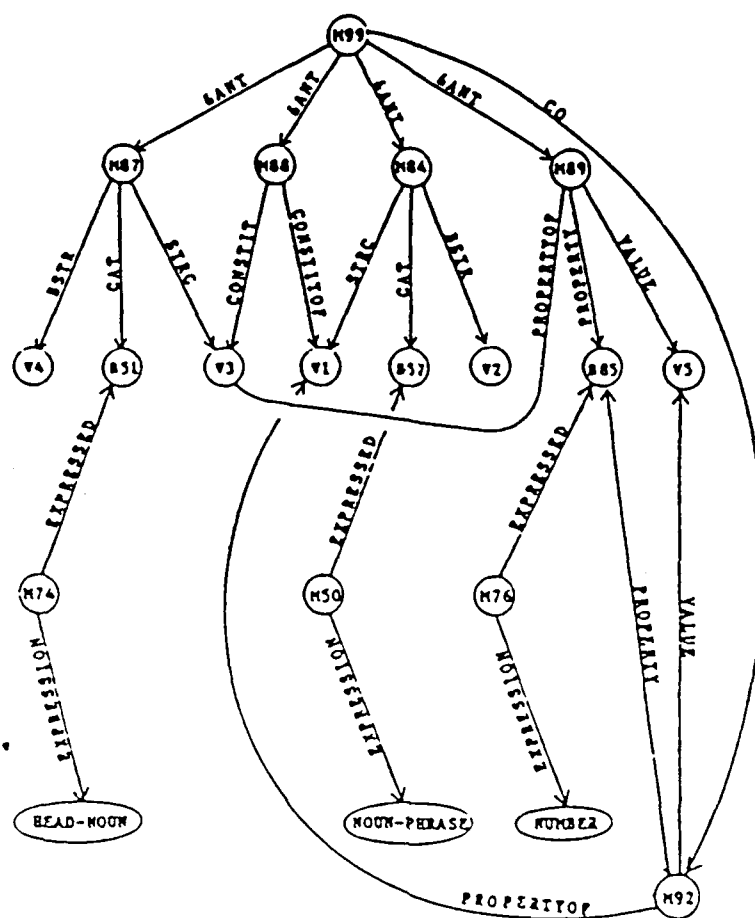


Figure 11-16: SNePS Net for Rule (R) (From Neal, 1985)

The representation of the following short narrative.

John arrived at the house. The sun was setting.
He rang the bell; a minute later, Mary opened
the door.

is shown in Figure 11-17. The ARG-PRED-EVENT case frame asserts that the proposition consisting of the argument pointed to by the ARG-arc and the predicate pointed to by the PRED-arc describes the event pointed to by the EVENT-arc. Notice that the predicates are classified into various types. This information plays an important role in the temporal analysis of a text.

NOW is a reference point that indicates the present moment of the narrative; it is updated as the story progresses through time. NOW is implemented as a variable whose current value is indicated in Figure 11-17 by a dotted arrow

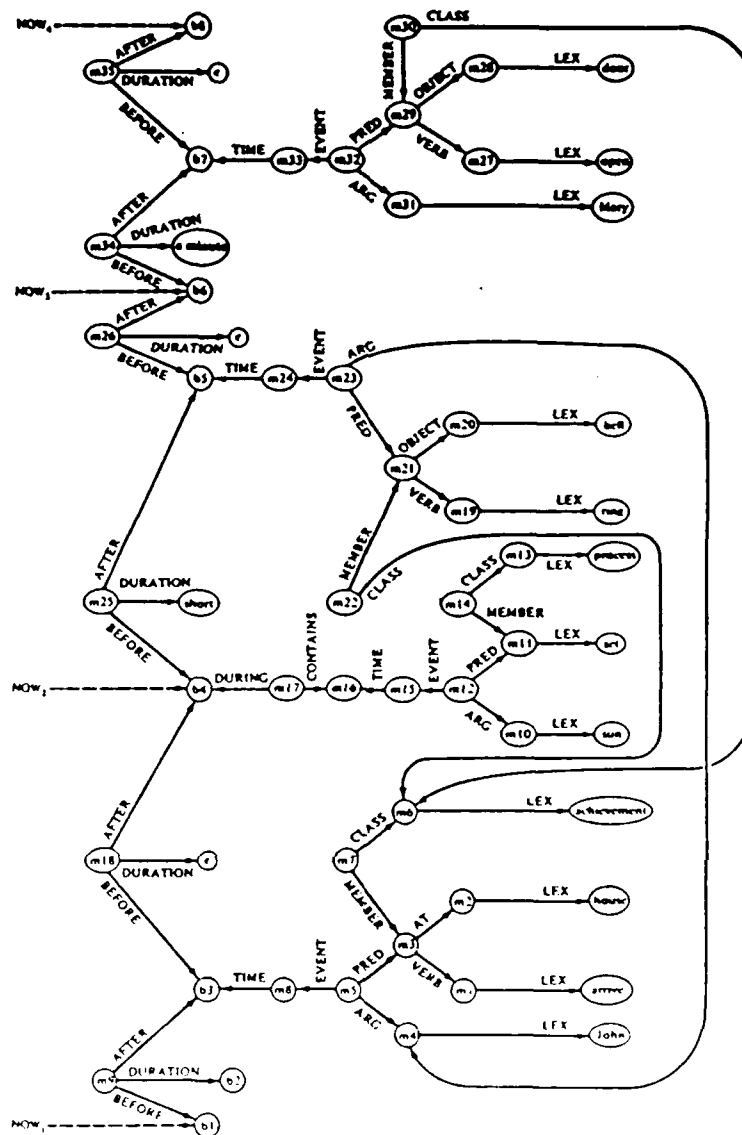


Figure 11-17: SNePS Network for a Short Narrative.

Subscripts are used in the figure to show the successive values of NOW.

The BEFORE-AFTER-DURATION case frame is used to indicate that the period of time pointed to by the BEFORE-arc temporally precedes the period of time pointed to by the AFTER-arc by the length of time pointed to by the DURATION-arc. These durations are usually not known precisely. The value <epsilon> stands for a very short interval; whenever an event occurs in the narrative line, it has the effect of moving NOW an interval of <epsilon> beyond it.

The DURING-CONTAINS case frame is used to indicate that the period of time pointed to by the DURING-arc is during (or contained in) the period of time pointed to by the CONTAINS-arc. Notice that the progressive sentence, "The sun was setting", created an event that contains the then-current NOW. If the system knows about such things as sunsets, then it should infer that the event of the sun's setting also contains John's arrival, his ringing of the bell, and probably also Mary's opening of the door.

11.6. Conclusion: SNePS and SNePS/CASSIE as Semantic Networks

We shall conclude by looking at SNePS and SNePS/CASSIE from the perspective of Brachman's discussions of structured inheritance networks such as KL-One and hierarchies of semantic network formalisms (Brachman, 1977, Brachman, 1979).

11.6.1. Criteria for semantic networks

Brachman offers six criteria for semantic networks:

A semantic network must have a *uniform notation*. SNePS provides some uniform notation with its built-in arc labels for rules, and it provides a uniform procedure for users to choose their own notation.

A semantic network must have an *algorithm for encoding information*. This is provided for by the interfaces to SNePS, for example, by the parser component of our ATN parser-generator that takes English sentences as input and produces SNePS networks as output.

A semantic network must have an *"assimilation" mechanism* for building new information in terms of stored information. SNePS provides for this by the Uniqueness Principle, which enforces node sharing during network building. The assimilation is demonstrated by the generator component of our ATN parser-generator, which takes SNePS nodes as input and produces English output expressing those nodes. Our conversation with CASSIE illustrated this: the node built to

represent the new fact. 'Lucy is sweet'. is expressed in terms of the already existing node for Lucy (who had previously been described as young) by 'young Lucy is sweet'.

A semantic network should be *neutral* with respect to network formalisms at higher levels in the Brachman hierarchy. SNePS is a semantic network at the "logical" level, whereas SNePS/CASSIE is at the "conceptual" level. SNePS is neutral in the relevant sense; it is not so clear whether SNePS/CASSIE is. But neutrality at higher levels may not be so important; a more important issue is the reasons why one formalism should be chosen over another. Several possible criteria that a researcher might consider are: *efficiency* (including the ease of interfacing with other modules; for example, our ATN parser-generator has been designed for direct interfacing with SNePS), *psychological adequacy* (irrelevant for SNePS, but precisely what SNePS/CASSIE is: being designed for), *ontological adequacy* (irrelevant for SNePS/CASSIE-see below), *logical adequacy* (guaranteed for SNePS, because of its inference package), and *natural language adequacy* (a feature of SNePS's interface with the ATN grammar).

A semantic network should be *adequate* for any higher-level network formalism. SNePS meets this nicely: KL-One can be implemented in SNePS (Tranch, 1982).

A semantic network should have a *semantics*. We presented that in Section 11.3. But it should be observed that there are at least two very different sorts of semantics. In SNePS, nodes have a meaning *within the system* in terms of their links to other nodes; they have a meaning *for users* as provided by the nodes at the heads of LEX arcs. Arcs, on the other hand, only have meaning within the system, provided by node- and path-based inference rules (which can be thought of as procedures that operate on the arcs). In both cases, there is an "internal", system semantics that is holistic and structural: the meaning of the nodes and arcs are not given in isolation, but in terms of the entire network. This sort of "syntactic" semantics differs from a semantics that provides links to an external interpreting system, such as a user or the "world" - that is, links between the network's way of representing information and the user's way. It is the latter sort of semantics that we provided for SNePS/CASSIE with respect to

an ontology of Meinongian objects.

11.6.2. SNePS and SNePS/CASSIE vs. KL-One

SNePS and SNePS/CASSIE can be compared directly to KL-One. Unlike KL-One, which is an *inheritance-network* formalism for representing concepts, instances of concepts, and properties and relations among them, SNePS is a *propositional-network* formalism for representing propositions and their constituents (individuals, properties, and relations).

Nevertheless, SNePS can handle inheritance. We have already seen an example of inheritance by *path*-based inference in the conversation with CASSIE. In that example, inheritance could also have been accomplished through node-based inference by, for example, representing 'dogs are animals' as a universally-quantified rule rather than by a SUBCLASS-SUPERCLASS case frame. That is, where an inheritance network might express the claim that dogs are animals by a single arc (say, a subclass-arc) from a dog-node to an animal-node, SNePS could express it by a proposition (represented by node m17 in Figure 11-5.).

One advantage of the propositional mode of representation and, consequently, of the second, or *rule*-based, form of property inheritance is that the proposition (m17) expressing the relationship can then become the objective of a proposition representing an agent's belief or it can become the antecedent or consequent of a node-based rule. In some inheritance networks, this could only be done by choosing to represent the entire claim by either the dog-node, the animal-node, the subclass-arc, or (perhaps) the entire structure consisting of the two nodes and the arc. The first two options seem incorrect; the third and fourth either introduce an anomaly into the representation (since arcs can then point either to nodes or to other arcs or to structures), or it reduces to what SNePS does: SNePS, in effect, trades in the single arc for a node with two outgoing arcs. In this way, the arcs of inheritance networks become information-bearing nodes, and the semantic network system becomes a propositional one.

Second, KL-One uses "epistemologically primitive links".

But why does KL-One use the particular set of links that it does, and not some other set: that is, what is the ontological justification for KL-One's links? There have been many philosophical and logical theories of the relations of the One to the Many (part-whole, member-set-superset, instance-concept, individual-species-genus, object-Platonic Form, etc.). KL-One's only motivation seems to be as a computationally efficient theory that clarifies the nature of inheritance networks; but it does not pretend to ontological or psychological adequacy. Indeed, it raises almost as many questions as it hopes to answer. For example, in KL-One, instances of a general concept seem to consist of *instances* of the attributes of the general concept, each of which instances have *instances* of the values of those attributes. But this begs important philosophical questions about the relations between properties of concepts (or of Forms, or of ...) and properties of individuals falling under those concepts (or participating in those Forms, or ...; some of these issues are discussed in (Brachman, 1983), but not from a philosophical point of view): Are they the same properties? Are the latter "instances" of the former? Are there such things as concepts (or Forms, or ...) of properties? And do instance nodes represent individuals? Do they represent individual concepts? [cf. (Brachman, 1977): 148.]

Now, on the one hand, SNePS/CASSIE's arcs are also taken to be "primitive"; but they are justified by the Meinongian philosophy of mind briefly sketched out above and explored in depth in the references cited. On the other hand, SNePS's arcs, by contrast to both SNePS/CASSIE's and KL-One's, are not restricted to any particular set of primitives. We believe that the interpretation of a particular use of SNePS depends on the *user's* world-view; the user should not be required to conform to *ours*.

And, unlike KL-One, the entities in the ontology for SNePS/CASSIE are not to be taken as representing things in the world: SNePS/CASSIE's ontology is an *epistemological ontology* [cf. (Rapaport, 1985a), (Rapaport, 1985b), (Rapaport, 1986a)] consisting of the purely intensional items that enable a

cognitive agent to have beliefs (about the world). An epistemological ontology is a theory of what there must be in order for a cognitive agent to have beliefs (about what there is).

References

- Abadir, M.S. and Breuer, M.A. A Knowledge-Based System for Designing Testable VLSI Chips. *IEEE Design and Test*, August 1985, 2(4), 56-68.
- Abelson, R. Concepts for Representing Mundane Reality in Plans. In Bobrow, D., and Collins, A. (Eds.), *Representation and Understanding*, New York: Academic Press, 1975.
- Adolph, W.S., Reghbati, H.K. and Sanmugasunderam, A. A Frame-Based System for Representing Knowledge About VLSI Design, pages . ACM-IEEE, 1986.
- Aho, A., Hopcroft, J., and Ullman, J. *Data Structures and Algorithms*. Reading, Massachusetts: Addison-Wesley, 1983.
- Aiello, N. *A Comparative Study of Control Strategies for Expert Systems: AGE Implementation of Three Variations of PUFF*. AAAI, Washington, 1983.
- Allen, J. *Maintaining Knowledge about Temporal Intervals*. Technical Report TR-86, Department of Computer Science, University of Rochester, 1981.
- Allen, J. Recognizing Intentions from Natural Language Utterances. In M. Brady (Ed.), *Computational Models of Discourse*, Cambridge, Massachusetts: M.I.T. Press, 1982.
- Allen, J. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 1983, 26(11), 832-843.
- Allen, J., and Kautz, H. A model for naive temporal reasoning. In Hobbs, J. and Moore, R. (Ed.), *Formal Theories of the Commonsense World*, Norwood, New Jersey: Ablex, 1985.
- Almeida, Michael J. *Reasoning about the Temporal Structure of Narrative Texts*. Technical Report 86-00, SUNY Buffalo Department of Computer Science, 1986.

- Almeida, Michael J., and Shapiro, Stuart C. *Reasoning about the Temporal Structure of Narrative Texts*. Cognitive Science Society, University of Rochester, 1983.
- Appelt, D. *Planning Natural Language Utterances to Satisfy Multiple Goals*. Technical Report TR, SRI International, 1982.
- Ari, M. Ben, Manna, Z., and Pnueli, A. *The Temporal Logic of Branching Time*, pages 222-235. Eight Annual ACM Symposium Principles of Programming Languages. , 1981.
- Bailes, A. *Response Generation*. Technical Report M.Sc. thesis, Department of Computing Science, University of Alberta, 1986.
- Baldwin, J.F. and Zhou, S.Q. *A Fuzzy Relational Inference Language*. Technical Report EM/FS132, University of Bristol, 1982.
- Barr, A., and Feigenbaum, E. *The Handbook of Artificial Intelligence: Volume 1*. Stanford, California:Harris Tech Press, 1981.
- Barrow, H., and Tenenbaum, J. *Representation and the Use of Knowledge in Vision*. Technical Report TR 108, SRI International, 1975.
- Barrow, H., and Tenenbaum, J. Recovering Intrinsic Scene Characteristics from Images. In A. Hanson and E. Riseman (Eds.), *Computer Vision Systems*. New York: Academic Press, 1978.
- Bartlett, F. *Remembering: A Study in Experimental and Special Psychology*. Cambridge, England:Cambridge University Press, 1932.
- Bellman, R.E. and Zadeh, L.A. Local and Fuzzy Logics. In Epstein, G. (Ed.), *Modern Uses of Multiple Valued Logic*, Dordrecht: Reidel, 1977.
- Birren, F. (ed). *Ostwald, The Color Primer*. New York, New York:Van Nostrand Reinhold Co., 1969.

- Birren, F. (ed). *Munsell's Grammar of Color*. New York. New York:Van Nostrand Reinhold Co., 1969.
- Black, F. A Deductive Question Answering System. In Minsky, M. (Eds.), *Semantic Information Processing*, Cambridge, Mass.: MIT Press, 1968.
- Blum, R. Discovery and Representation of Causal Relationships from a Large Time-Oriented Clinical Database. In *Lecture Notes in Medical Informatics 19*, Springer-Verlag, 1982.
- Bobrow, D. Dimensions of Representation. In Bobrow, D., and Collins, A. (Eds.), *Representation and Understanding*, New York: Academic Press, 1975.
- Bobrow, D. *Special Issue on Non-Monotonic Logic of Artificial Intelligence Journal*. Amsterdam:North Holland, 1980.
- Bobrow, R., and Webber, B. *Knowledge Representation for Syntactic/Semantic Processing*, pages 316-323. Proceedings of the 1st AAAI, Stanford, California, 1980.
- Bobrow, D., and Winograd, T. An Overview of KRL: a Knowledge Representation Language. *Cognitive Science*, 1977, 1(1), 3-46.
- Boland, J., and Lekkerkerker, C. Representation of a Finite Graph by a Set of Intervals on the Real Line. *Fundamentals Mathematics*, 1962, 11(1), 45-64.
- Booth, K., and Lueker, G. *Linear Algorithms to Recognize Interval Graphs and Test for the Consecutive Ones Property*, pages 252-265. Proceedings of the 7th ACM Symposium on the Theory of Computing, . 1975.
- Brachman, Ronald J. What's in a Concept: Structural Foundations for Semantic Networks. *International Journal of Man-Machine Studies*, 1977, 9, 127-52.
- Brachman, Ronald J. On the Epistemological Status of Semantic Networks. In Findler, N.V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, New York: Academic Press, 1979. reprinted in Brachman and

Levesque 1985: 191-215.

Brachman. R. *What IS-A is and Isn't*. CSCSI, Saskatoon, 1982.

Brachman. R. What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. *IEEE Computer*, 1983, 16(10), 30-36.

Brachman. R., and Levesque. H. *Readings in Knowledge Representation*. Los Altos, Ca.:Morgan Kaufmann, 1985.

Brachman. R., and Schmolze. J. An Overview of the KL-One Knowledge Representation System. *Cognitive Science*, 1985, 9(4), 171-216.

Brachman. R., and Smith. B. *Special Issue on Knowledge Representation*, *SIGART Newsletter*. New York:ACM, 1980.

Brachman. R., Fikes. R., and Levesque. H. KRYPTON: A Functional Approach to Knowledge Representation. *IEEE Computer*, 1983, 16(10), 67-74.

Brachman. Ronald J.; Gilbert. Victoria P.; and Levesque. Hector J. *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON*, pages 532-39. IJCAI, 1985.

Brower. R., Meester. G. *The Shape of the Human Left Ventricle: Quantification of Symmetry*. Computers in Cardiology, Florence, 1981.

Brown. J. S., and Burton. R. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, 1978, 2(2), 155-192.

Brown. J., Burton. R., and Bell. A. *SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI)*. Technical Report BBN Report No. 2790, Bolt Beranek and Newman, Inc., 1974.

Brown. J., Burton. R., and de Kleer. J. Pedagogical, Natural Language and Knowledge Engineering Techniques in

SOPHIE I. II. and III. In Sleeman, D., and Brown, J. (Eds.), *Intelligent Tutoring Systems*, New York: Academic Press, 1982.

Buchanan, B., Sutherland, G., and Feigenbaum, E. Heuristic DENDRAL: A Program for Generating Explanatory Hypothesis in Organic Chemistry. In B. Meltzer and D. Michie (Eds.), *Machine Intelligence 4*, Edinburgh: Edinburgh University Press, 1969.

Buchanan, B., and Shortliffe, E. Rule Based Expert Systems: The Mycin Experiments of the Stanford HPP. In Buchanan, B., and Shortliffe, E. (Eds.), *Rule Based Expert Systems: The Mycin Experiments of the Stanford HPP*, Reading, Massachusetts: Addison-Wesley, 1984.

Bundy, A., and Silver, B. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 1981, 16(2), .

Bundy, A., Byrd, L., and Mellish, C. *Special purpose, but domain independent, inference mechanisms.*, pages 67-74. AISB, Orsay, France, 1982.

Carbonell, J. POLITICS. In R. Schank and C. Reisbeck (Eds.), *Inside Computer Understanding*, Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1981.

Carbonell, J., Michalski, J., and Mitchell, T. An Overview of Machine Learning. In Michalski, J., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, California: Tioga Press, 1983.

Carnap, Rudolf. *The Logical Structure of the World*. Berkeley: University of California Press, 1967. R.A. George, translator.

Castañ, Hector-Neri. Thinking and the Structure of the World. *Philosophia*, 1974, 4, 3-40. reprinted in 1975 in *Critica* 6(1972)43-86.

Castañeda, Hector-Neri. Individuals and Non-Identity: A New

- Look *American Philosophical Quarterly*. 1975. 12. 131-40.
- Castañeda, Hector-Neri. Identity and Sameness. *Philosophia*. 1975. 5. 121-50.
- Castañeda, Hector-Neri. Perception, Belief, and the Structure of Physical Objects and Consciousness. *Synthese*. 1977. 35. 285-351.
- Castañeda, Hector-Neri. Fiction and Reality: Their Basic Connections. *Poetica*. 1979. 8. 31-62.
- Castañeda, Hector-Neri. *Thinking and Doing: The Philosophical Foundations of Institutions*. Dordrecht:Reidel, 1977.
- Cercone, N., and McCalla, G. Artificial Intelligence: Underlying Assumptions and Basic Objectives. *American Journal for Information Science*. 1984. 5(35). 280-290.
- Cercone, N., and McCalla, G. Accessing Knowledge Through Natural Language. In M. Yovits (Ed.), *Advances in Computers*. New York: Academic Press. 1986.
- Cercone, N. and Schubert, L. *Toward a State-Based Conceptual Representation*, pages 83-91. IJCAI75. Advance Papers of the 4th International Joint Conference on Artificial Intelligence, Tbilisi, USSR. 1975.
- Cercone, N., McCalla, G., and McFetridge, P. The Many Dimensions of Logical Form. In Bolc, L. (Eds.), *Translating Natural Language into Logical Form*. New York: Springer-Verlag. 1986. forthcoming.
- Chandrasekaran, B., Gomez, F., Mittal, S., Smith, J. *An Approach to Medical Diagnosis Based on Conceptual Structures*. IJCAI. Tokyo. 1979.
- Chapman, David. *Planning for Conjunctive Goals*. Technical Report 802. MIT Artificial Intelligence Laboratory. November 1985.
- Cheng, M. *The Design and Implementation of the Waterloo UNIX Prolog Environment*. Technical Report Report 26. CS-84-47, University of Waterloo. 1984.

- Clancey. W. Tutoring Rules for Guiding a Case Method Dialogue. In Sleeman, D., and Brown, J. (Eds.). *Intelligent Tutoring Systems*, London, England: Academic Press, 1982.
- Clark, K., and McCabe, F. The control facilities of IC-PROLOG. In Michie, D. (Eds.). *Expert systems in the micro-electronic age*, Edinburgh, Scotland: Edinburgh University Press, 1979.
- Clocksin, W., and Mellish, C. *Programming in Prolog*. Berlin:Springer-Verlag, 1981.
- Codd, E. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 1970, 13(6), 377-387.
- Cohen, P. *Planning Speech Acts*. Technical Report TR 118, Department of Computer Science, University of Toronto, 1978.
- Collins, A. Reasoning from Incomplete Knowledge In Bobrow, D., and Collins, A. (Eds.). *Representation and Understanding*. New York: Academic Press, 1975.
- Collins, A., and Quillian, M. How to Make a Language User. In Tulving, E., and Donaldson, W. (Eds.). *Organisation of Memory*. New York: Academic Press, 1972.
- Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, Ph. *Un Systeme de Communication Homme-Machine en Francais*. Marseille, France: Aix-Marseille University Press, 1973.
- Colmerauer, A. *Un Systeme de Communication Homme-Machine en Francais*. Technical Report, Aix-Marseille University, 1973.
- Covington, A. and Schubert, L. *Organization of modally embedded propositions and of dependent concepts*, pages 87-94. CSCSI/SCEIO, Victoria, British Columbia, 1980.
- Craig, J., et al. DEACON: Direct English Access and Control. *FJCC, AFIPS Conf. Proc.*, 1966, 29(1), 365-380.
- Cresswell, M.J. *Logic and Languages*. London: Methuen, 1973.

- Cullingford, R. *Script Application: Computer Understanding of Newspaper Stories*. Technical Report Research Report 116. Department of Computer Science. Yale University. 1978.
- Dahl, V. Translating Spanish into Logic Through Logic. *American Journal of Computational Linguistics*. 1981. 7(3). x-y.
- Dahl, V. On Database Systems Development Through Logic. *ACM Transactions on Database Systems*. 1982. 7(1). 102-123.
- Dahl, V., and McCord, M. *Treating Coordination in Logic Grammars*. Technical Report TR-83. Simon Fraser University and University of Kentucky. 1983.
- Dahl, V., and Sambuc, R. *Un Systeme de Banque de Donnees en Logique du Premier Ordre, en Vue de sa Consultation en Langue Naturelle*. Technical Report . Aix-Marseille University. 1976.
- Date, C.J. *An Introduction to Database Systems*. Reading: Addison-Wesley. 1977.
- Date, C.J. *An Introduction to Database Systems*. Reading, Massachusetts: Addison-Wesley. 1981.
- Davis, R. Diagnostic Reasoning Based on Structure and Behaviour. *Artificial Intelligence*. 1984. 24(4). 347-410.
- Davis, R., Buchanan, B., and Shortliffe, E. Production Rules as a Representation for a Knowledge-Based Consultation Program. *Artificial Intelligence*. 1977. 8(1). 15-45.
- de Groot, A. Perception and Memory Versus Thought: Some Old Ideas and Recent Findings. In Kleinmuntz, B. (Eds.), *Problem Solving*, New York: Wiley. 1967.
- De Kleer, J., and Brown, J. A Qualitative Physics Based on Confluence. *Artificial Intelligence*. 1984. 24(2). 7-83.
- Dean, Thomas. *Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving*. Technical Report YALEU/CSD/RR #433. Yale University.

Department of Computer Science, October 1985.

deHaan, J. *Inference in a topically organized semantic net.*
Technical Report M.Sc. thesis. Department of Computing
Science, University of Alberta, 1986.

deHaan, J., and Schubert, L. *Inference in a Topically Organized
Semantic Net.*, pages (to appear). AAAI, Philadelphia,
Pennsylvania, 1986.

DeKleer, J. How Circuits Work. *Artificial Intelligence*, 1984,
24, 205-280.

Delgrande, J. *Steps Towards a Theory of Exceptions*, pages
87-94. Proceedings of the 5th National Conference of the
CSCSI/SCEIO, London, Ontario, 1984.

Dennett, D. Why the Law of Effect will not go away. In
(Eds.), *Brainstorms*, Cambridge, Massachusetts: Bradford
Books, MIT Press, 1978.

Doran, J., Traill, T., Brown, D., Gibson, D. Detection of
Abnormal Left Ventricular Wall Movement During
Isovolumic Contraction and Early Relaxation. *British
Heart Journal*, 1978, 40, .

Doyle, J. *Some Theories of Reasoned Assumptions: An Essay in
Rational Psychology*. Technical Report TR, Carnegie
Mellon University, 1982.

Doyle, J. *A Society of Mind: Multiple Perspectives, Reasoned
Assumptions, and Virtual Copies*, pages . Eight
International Joint Conference on Artificial Intelligence,
Karlsruhe, West Germany, 1983.

Dresher, B., and Hornstein, N. On Some Supposed
Contributions of Artificial Intelligence to the Scientific
Study of Language. *Cognition*, 1976, 4(4), 321-398.

Dresher, B., and Hornstein, N. Reply to Winograd. *Cognition*,
1976, 5(4), 377-392.

Dresher, B., and Hornstein, N. Response to Schank and
Wilensky. *Cognition*, 1977, 5(2), 147-150.

- DuBois, D. and Prade, H. Fuzzy Cardinality and the Modelling of Imprecise Quantification. *Fuzzy Sets and Systems*, 1985, 16, 199-230.
- Duda, R., Gashning, J., Hart, P., Konolige, K., Reboh, R., Barrett, P., and Slocum, J. *Development of the PROSPECTOR Consultation System for Mineral Exploration, Final Report, SRI Projects 5821 and 6415*. Technical Report 5821 & 6415, SRI International, 1978.
- Elcock, E. Problem solving compilers. In Findler, N. (Eds.), *Artificial Intelligence and Heuristic Programming*, Edinburgh, Scotland: Edinburgh University Press, 1971.
- Elcock, E., McGregor, J., and Murray, A. Data directed control and operating systems. *British Computer Journal*, 1972, 15(2), 125-129.
- Etherington, D., Mercer, R., and Reiter, R. On the Adequacy of Predicate Circumscription for Closed Reasoning. *Computational Intelligence*, 1985, 1(1), 11-15.
- Fagan, L. M. *VM: Representing Time Dependence Relations in a Medical Setting*. Technical Report Ph.D. Thesis, AI Laboratory, 1980.
- Fahlman, S. *A System for Representing and Using Real World Knowledge*. Technical Report AI Lab Memo 331, Project MAC, M.I.T., 1975.
- Fahlman, S. *NETL: A System for Representing and Using Real World Knowledge*. Cambridge, Mass.:M.I.T. Press, 1979.
- Fahlman, S. *Design Sketch for a Million Element NETL Machine*, pages 249-252. Proceedings of the 1st AAAI, Stanford, California, 1980.
- Fahlman, S. *Three Flavors of Parallelism*, pages 230-235. CSCSI 82, Proceedings of the 4th National Conference of the Canadian Society for Computational Studies of Intelligence, Saskatoon, Sask., 1982.
- Fikes, Richard E. and Nilsson, Nils J. STRIPS: A New

- Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 1971, (2)}, 198-208.
- Fikes, R., and Nilsson, N. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 1971, 2(3), 184-208.
- Findlay, J.N. *Meinong's Theory of Objects and Values*. Oxford:Clarendon Press, 1963.
- Fine, Kit. *A Defence of Arbitrary Objects*, pages 55-77. Aristotelian Society, 1983. Supp. Vol. 58.
- Firby, R. James, Dean, Thomas and Miller, David. *Efficient Robot Planning with Deadlines and Travel Time*. IASTED, Santa Barbara, Ca., May, 1985.
- Fodor, J. Tom Swift and His Procedural Grandmother. *Cognition*, 1978, 6(4), 229-247.
- Forbus, K. Qualitative Process Theory. *Artificial Intelligence*, 1984, 24(2), 85-168.
- Forgy, C., and McDermott, J. OPS, A Domain Independent Production System Language, pages 933-939. IJCAI77. Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Mass., 1977.
- Foster, J., and Elcock, E. Absys 1: an incremental compiler for assertions: an introduction. In Meltzer, B., and Michie, D. (Eds.), *Machine Intelligence 4*, Edinburgh, Scotland: Edinburgh University Press, 1969.
- Fujii, J., Watanabe, H., Koyama, S., Kato, K. Echocardiographic Study on Diastolic Posterior Wall Movement Left Ventricular Filling by Disease Category. *American Heart Journal*, 1979, 98, .
- Fulkerson, D., and Gross, O. Incidence Matricies and Interval Graphs. *Pacific Journal of Mathematics*, 1965, 15(11), 835-855.
- Funt, B. Problem Solving with Diagrammatic Representations

- Artificial Intelligence*. 1980. 13(3), 201-230.
- Funt, B. A Parallel Process Model of Mental Rotation. *Cognitive Science*. 1983, 4(1), 1-23.
- Gerbrands, J., Booman, F., Reiber, J. Computer Analysis of Moving Radiopaque Markers from X-Ray Films. *Computer Graphics and Image Processing*. 1979, 11, .
- Gershon, R. Explanation Methods for Visual Motion Understanding Systems. Master's thesis. Department of Computer Science, University of Toronto, 1982.
- Ghosh, S. File Organization: the Consecutive Retrieval Property. *Communications of the ACM*. 1972, 15(12), 802-808.
- Ghosh, S., Kambayashi, Y., and Lipski, W. Data Base File Organization, Theory and Applications of the Consecutive Retrieval Property. In Ghosh, S., Kambayashi, Y., and Lipski, W. (Eds.), *Data Base Organization*, New York: Academic Press, 1983.
- Gibson D., Prewitt, T., Brown, D. Analysis of Left Ventricular Wall Movement During Isovolumic Relaxation and its Relation to Coronary Artery Disease. *British Heart Journal*. 1976, 38, .
- Glicksman, J. A Schemata-Based System for Utilizing Cooperating Knowledge. pages 33-39. Proceedings of the 4th National Conference of the CSCSI/SCEIO, Saskatoon, 1982.
- Goebel, R. *Interpreting Descriptions in a Prolog Based Knowledge Representation System*, pages 711-716. IJCAI 85, Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, California, 1985.
- Goldstein, I., and Burton, R. The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. In Sleeman, D., and Brown, J. (Eds.), *Intelligent Tutoring Systems*, New York: Academic Press, 1982.
- Goodwin, James. *Taxonomic Programming with KL-One*.

- Green, C. *A Summary of the PSI Program Synthesis System*, pages 380-381. IJCAI77. Proceedings of the 4th International Joint Conference on Artificial Intelligence. Cambridge, Massachusetts. 1977.
- Hadley, R. SHADOW: A Natural Language Query Analyser. *Computers and Mathematics*. 1985. 11(5). x-y.
- Hagan, A., et al. Evaluation of Computer Programs for Clinical Electrocardiography. In D. Cady, Jr. (Ed.), *Computer Techniques in Cardiology*, New York: Marcel Dekker, Inc., 1979.
- Halmos, P. *Lectures on Boolean Algebra*. New York: Van Nostrand Press. 1963.
- Hanson, A., and Riseman, E. VISIONS: A Computer Systems for Interpreting Scenes. In A. Hanson and E. Riseman (Eds.), *Computer Vision Systems*, New York: Academic Press. 1978.
- Havens, W. *A Procedural Model of Recognition*, pages 263-264. Proceedings of the 5th IJCAI. MIT. Cambridge, MA. 1977.
- Havens, W. *A Procedural Model of Recognition for Machine Perception*, pages 254-262. Proceedings of the 2nd National Conference of the CSCSI/SCEIO, Toronto. 1978.
- Havens, W. Recognition Mechanisms for Hierarchical Schemata Knowledge. *Computers and Mathematics*. 1983. 9(1), 185-200.
- Havens, W. A Theory of Schema Labelling. *Computational Intelligence*. 1985. 1(3). 101-120.
- Hawrylak, I. M.Sc. Thesis . Master's thesis. Department of Computer Science. University of Kentucky. 1985.
- Hayes, P. *In Defence of Logic*, pages 559-565. IJCAI77. Proceedings of the 5th International Joint Conference on Artificial intelligence. Cambridge, Mass., 1977.

- Hayes. P. The Logic of Frames. In B. Webber and N. Nilsson (Eds.), *Readings in Artificial Intelligence*, Palo Alto, California: Tioga Press, 1979.
- Hayes-Roth, F. Using Proofs and Refutations to Learn from Experience. In Michalski, R., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, California: Tioga Press, 1983.
- Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., (Editors). *Building Expert Systems*. Reading, Mass.:Addison-Wesley, 1983.
- Hendrix, G. *Expanding the Utility of Semantic Networks Through Partitioning*, pages 115-121. IJCAI75, Advance Papers of the 4th International Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975.
- Hendrix, G. Encoding Knowledge in Partitioned Networks. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*, New York: Academic Press, 1979.
- Hewitt, C. *Description and Theoretical Analysis of PLANNER*. Technical Report Ph.D. Thesis, AI Laboratory, M.I.T., 1972.
- Hobbs, J.R. and Moore, R.C. *Formal Theories of the Commonsense World*. Norwood:Ablex, 1984.
- Hobbs, J.R. and Moore, R.C. *Formal Theories of the Commonsense World*. Norwood, NJ:Ablex, 1985.
- Hoehne, K., Boehm, M., Nicolae, G. The Processing of X-Ray Image Sequences. In Stucki (Ed.), *Advances in Digital Image Processing*, Plenum Press, 1980.
- Horowitz, S. A Syntatic Algorithm for Peak Detection in Waveforms with Applications to Cardiography. *Communications of the ACM*, May 1975, 18(5).
- Horstmann, P.W. *A Knowledge-Based System Using Design for Testability Rules*, pages 278-284. Proceedings of FTCS-14.

1984.

Israel, David J. Interpreting Network Formalisms. In Cercone, Nick (Ed.), *Computational Linguistics*. Oxford: Pergamon Press, 1983.

Johnson-Laird, P. Procedural Semantics. *Cognition*, 1977, 5, 189-214.

Johnson-Laird, P. Mental Models in Cognitive Science. *Cognitive Science*, 1980, 4(1), 71-115.

Jones, M., and Poole, D. *An Expert System for Educational Diagnosis based on Default Logic*, pages 573-583. Proceedings of the 5th International Workshop on Expert Systems and their Applications, Avignon, France, 1985.

Joobani, R. and Siewiorek, D.P. *Weaver: A Knowledge-Based Routing Expert*, pages 266-272. ACM-IEEE, 1985.

Judd, D., and Wysecki, G. *Color in Business, Science and Industry (2nd edition)*. New York, New York: John Wiley and Sons, 1963.

Kahn, K., and Gorry, G. Mechanizing Temporal Knowledge. *Artificial Intelligence*, 1977, 9(2), 87-108.

Kameda, T. On the Vector Representation of the Reachability in Planar Directed Graphs. *Information Processing Letters*, 1975, 3(4), 75-77.

Kao, M. Turning Null Responses into Quality Responses. Master's thesis, School of Computing Science, Simon Fraser University, 1986.

Kaplan, J. Cooperative Responses from a Portable Natural Language Query System. *Artificial Intelligence*, 1982, 19(2), 165-187.

Katz, R.H. Managing the Chip Design Database. *IEEE Computer*, December 1983, 16(12), 26-40.

Kaufmann, A. and Gupta, M. *Introduction to Fuzzy Arithmetic*. New York: VanNostrand, 1985.

- Kay, P. *Color Perception and the Meanings of Color Words*. pages 61-64. Cog. Sci. Soc., Berkeley, California, 1981.
- Kosslyn, S. *Image and Mind*. Cambridge, Massachusetts:Harvard University Press, 1980.
- Kowalski, R. *Predicate Logic as a Programming Language*, pages 569-574. North Holland Publishing Company, Amsterdam, 1974.
- Kowalski, R. *Logic for Problem Solving*. New York, New York:North Holland Elsevier, 1979.
- Kowalski, T.J., et al. The VLSI Design Automation Assistant: From Algorithms to Silicon. *IEEE Design and Test*, August 1985, 2(4), 33-34.
- Kuipers, B. Commonsense Reasoning about Causality: Deriving Behaviour from Structure. *Artificial Intelligence*, 1984, 24(2), 169-202.
- Kunz, J.C. *Analysis of Physiological Behavior using a Causal Model based on First Principles*. American Association for Artificial Intelligence, August, 1983.
- Kuratowski, K., and Mostowski, A. *Set Theory*. New York:North Holland, 1982.
- Lambert, Karel. *Meinong and the Principle of Independence*. Cambridge, England:Cambridge University Press, 1983.
- Langley, P., Bradshaw, G., and Simon, H. Rediscovering Chemistry with the Bacon System. In Michalski, R., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, California: Tioga Press, 1983.
- Lenat, D. The Role of Heuristics in Learning by Discovery: Three Case Studies. In Michalski, R., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, California: Tioga Press, 1983.
- Levesque, H. *A Procedural Approach to Semantic Networks*.

Technical Report TR-105, Department of Computer Science, University of Toronto, 1977.

Levesque, H. *A Formal Treatment of Incomplete Knowledge Bases*. Technical Report Ph.D. Thesis, Department of Computer Science, University of Toronto, 1981.

Levesque, Hector J., and Brachman, Ronald J. A Fundamental Tradeoff in Knowledge Representation and Reasoning. In *Brachman and Levesque 1985*, Morgan Kaufmann, 1985. Revised Version.

Levesque, H., and Mylopoulos, J. A Procedural Semantics for Semantic Networks. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*, New York: Academic Press, 1979.

Lispki, W. Information Storage and Retrieval. *Theoretical Computing Science*, 1976, 3(3), 183-211.

Lipski, W. *Logical Problems Related to Incomplete Information in Databases*. Technical Report Preprint #452, Universite de Paris-Sud, Orsay, 1977.

Long, W. *Reasoning about State from Causation and Time in a Medical Domain*. American Association for Artificial Intelligence, August, 1983.

Long, W., Russ, T., *A Control Structure for Tim-Dependent Reasoning*. IJCAI, Karlsruhe, Germany, 1983.

Loveland, D. *Automated Theorem Proving: A Logical Basis*. Amsterdam:North Holland, 1978.

Lukasiewicz, J. *Aristotle's Syllogistic*. Oxford:Clarendon Press, 1951.

Mackworth, A. Consistency in Networks of Relations. *Artificial Intelligence*, 1975, 8(1), 99-118.

Mackworth, A. Model Driven Interpretation in Intelligent Vision Systems. *Perception*, 1976, 5(), 349-370.

Mackworth, A. How to See a Simple World. In E. Flock,

and D. Michie (Eds.). *Machine Intelligence 8*, New York: Halstead Press. 1977.

Mackworth, A. Vision Research Strategy: Black Magic, Metaphors, Mechanisms, Miniworlds, and Maps. In A. Hanson and E. Riseman (Eds.). *Computer Vision Systems*, New York: Academic Press. 1978.

Mackworth, A. *Constraints, Descriptions and Domain Mappings in Computational Vision*, pages . Royal Society Symposium on Physical and Biological Processing of Images. London. 1983.

Mackworth, A., and Freuder, E. *The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems*. Technical Report TR 82-6. University of British Columbia. 1982.

Mackworth, A. and Havens, W. *Structuring Domain Knowledge for Visual Perception*, pages 625. IJCAI81, Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, Canada. 1981.

Maida, Anthony S. and Shapiro, Stuart C. Intensional Concepts in Propositional Semantic Networks. *Cognitive Science*, 1982, 6, 291-330. Reprinted in Brachman and Levesque 1985: 169-89.

Maier, D. *The Theory of Relational Databases*. New York:Computer Science Press. 1983.

Mamdani, E.H. and Gaines, B.R. *Fuzzy Reasoning and its Applications*. London:Academic Press. 1981.

Marr, D. *Vision*. San Francisco, Ca.:W.H. Freeman. 1982.

Martins, João and Shapiro, Stuart C. *Reasoning in Multiple Belief Spaces*, pages 370-73. IJCAI-8, 1983.

Martins, João. *Reasoning in Multiple Belief Spaces*. Technical Report 203, SUNY Buffalo Dept. of Computer Science. 1983.

Martins, João. Belief Revision. In Shapiro, S.C. (Ed.).

Encyclopedia of Artificial Intelligence. New York: John Wiley. 1987.

Martins, João and Shapiro, Stuart C. *A Model for Belief Revision*, pages 241-94. AAAI, 1984.

Martins, João and Shapiro, Stuart C. Theoretical Foundations for Belief Revision. In Halpern, J.Y. (Ed.), *Theoretical Aspects of Reasoning About Knowledge*, Los Altos, California: Morgan Kaufmann, 1986.

Martins, João and Shapiro, Stuart C. *Hypothetical Reasoning*, pages 1029-42. AAAI, Berlin, 1986.

Martins, João and Shapiro, Stuart C. *Belief Revision in SNePS*, pages 230-34. CSCSI, 1986.

Mathlab Group. *MACSYMA Reference Manual*. Cambridge, Massachusetts: Computer Science Laboratory, Massachusetts Institute of Technology, 1977.

Mays, E. *Correcting Misconceptions About Database Structure*, pages 123-128. Proceedings of the 3rd CSCSI National Conference, Victoria, British Columbia, 1980.

Mays, E. *Monitors as Responses to Questions: Determining Competence*, pages 421-423. Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, Pennsylvania, 1982.

McCalla, G. *An Approach to the Organisation of Knowledge for the Modelling of Conversation*. Technical Report PhD Thesis, University of British Columbia, 1978.

McCalla, G., and Cercone, N. Techniques and Issues in the Design of Applied Artificial Intelligence Systems. *Computers and Mathematics*, 1985, 11(5), 421-430.

McCarthy, J. First Order Theories of Individual Concepts and Propositions. In Hayes, J.E., D. Michie, and L. Mikulich (Ed.), *Machine Intelligence 9*, Chichester, England: Ellis Horwood, 1979. reprinted in Brachman and Levesque 1985: 523-533.

- McCarthy, J. Circumscription: A Form of Non-Monotonic Reasoning. *Artificial Intelligence*. 1980. 13(1.2). 27-39.
- McCarthy, J. *Applications of Circumscription to Formalizing Commonsense Knowledge*. Technical Report AI Technical Report. Stanford, November 1984.
- McCarthy, J., and Hayes, P. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B., and Michie, D. (Eds.), *Machine Intelligence*. New York: American Elsevier. 1969.
- McCarty, L. and Sridharan, N. *The Representation of an Evolving System of Legal Concepts II: Prototypes and Deformations*, pages 246-253. IJCAI81, Proceedings of the 7th International Joint Conference on Artificial Intelligence. Vancouver, Canada, 1981.
- McCoy, K. *Augmenting a Database Knowledge Representation for Natural Language Generation*, pages 121-128. Proceedings of the 20th ACL, Toronto, Ontario, 1982.
- McDermott, Drew. *The DUCK Manual*. Technical Report YALEU/CSD/RR #399. Yale University. Department of Computer Science, June 1985.
- McKay, Donald P., and Martins, João. *SNePSLOG User's Manual*. Technical Report SNeRG Technical Note #4. SUNY Buffalo Department of Computer Science, 1981.
- McKay, Donald P., and Shapiro, Stuart C. *MULTI - A LISP-Based Multiprocessing System*, pages 29-37. AAAI. Stanford University, 1980.
- McKeown, K. *The TEXT System for Natural Language Generation: An Overview*, pages 113-120. Proceedings of the 20th ACL, Toronto, Ontario, 1982.
- McSkimmin, J.R., and Minker, J. *The Use of a Semantic Network in a Deductive Question-Answering System*, pages 50-58. IJCAI77, Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Mass., 1977.

- McSkimmin, J.R., and Minker, J. A Predicate Calculus Based Semantic Network For Deductive Searching. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*. New York: Academic Press. 1979.
- Mead, C. and Conway, L. *Introduction to VLSI Systems*. Reading, Massachusetts: Addison-Wesley. 1980.
- Meinong, Alexius. *Über Gegenstandstheorie*. In Haller, R. (Ed.), *Alexius Meinong Gesamtausgabe, Vol. II*. Graz, Austria: Akademische Druck-u. Verlagsanstalt. 1904. English Translation *The Theory of Objects* by I. Levi et al., pp. 76-117 in R.M. Chisholm (ed.), *Realism and the Background of Phenomenology* (New York: Free Press. 1960).
- Meinong, Alexius. *On Assumptions*. Berkeley: University of California Press. 1983.
- Mercer, R., and Reiter, R. *The Representation of Presuppositions Using Defaults*, pages 103-107. Proceedings of the 4th National Conference of the CSCSI/SCEIO, Saskatoon, 1982.
- Miller, David P. *Planning by Search Through Simulations*. Technical Report YALEU/CSD/RR #423, Yale University, Department of Computer Science, October 1985.
- Minsky, M. A Framework for Representing Knowledge. In P. Winston (Ed.), *Psychology of Computer Vision*, New York: McGraw Hill, 1975.
- Minsky, M. *Learning Meaning*. Technical Report AI Lab Memo. Project MAC, M.I.T., 1980.
- Minsky, M. Why People Think Computers Can't. *Artificial Intelligence Magazine*, 1982, 3(4), 3-15.
- Moore, R. *Reasoning About Knowledge and Actions*. Technical Report Tech note 191, SRI International. 1980.
- Moore, R. *The Role of Logic in Knowledge Representation and Commonsense Reasoning*, pages 428-433. Proceedings of the 2nd AAAI, Pittsburgh, Pennsylvania. 1982.

- Mostow, D.J., Hayes-Roth, F. A Production System for Speech Understanding System. In Waterman and Hayes-Roth (Ed.). *Pattern Directed Inference Systems*, Academic Press. 1978.
- Mylopoulos, J., Shibahara, T., and Tsotsos, J. Building Knowledge-Based Systems: The PSN Experience. *IEEE Computer special issue on Knowledge Representation*, 1983, 16(10), 83-89.
- Neal, Jeanette G. *A Knowledge Based Approach to Natural Language Understanding*. Technical Report 85-06, SUNY Buffalo Department of Computer Science, 1985.
- Newell, A. Production Systems: Models of Control Structure. In W. Chase (Eds.). *Visual Information Processing*, New York: Academic Press, 1973.
- Newell, A. and Simon, H. *Human Problem Solving*. Englewood Cliffs, NJ:Prentice-Hall, 1972.
- Nii, H., and Aiello, N. AGE: A Knowledge-Based Program for Building Knowledge-Based Programs, pages 645-655. IJCAI79, Proceedings of the 6th International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.
- Nilsson, N. *Problem Solving Methods in Artificial Intelligence*. New York, New York:McGraw Hill, 1971.
- Nilsson, N. *Principles of Artificial Intelligence*. Palo Alto, California:Tioga Press, 1980.
- Nilsson, N. Artificial Intelligence: Engineering, Science, or Slogan? *Artificial Intelligence Magazine*, 1982, 3(1), 2-8.
- Noguchi, K., Umamo, M., Mizumoto, M., and Tanaka, K. *Implementation of Fuzzy Artificial Intelligence Language FLOU*. Technical Report, Univ of Tokyo, 1976. Technical Report on Automation and Language of IECE.
- Norman, D., and Rumelhart, D. *Explorations in Cognition*. San Francisco, Ca.:W.H. Freeman, 1975.
- Papalaskaris, M. *Special Purpose Inference Methods*. Technical

- Report M.Sc. thesis, Department of Computing Science.
University of Alberta, 1982.
- Papalaskaris, M., and Schubert, L. *Parts Inference: Closed and Semi-closed Partitioning Graphs*, pages 304-309. IJCAI, Vancouver, British Columbia, 1981.
- Papalaskaris, M., and Schubert, L. *Inference, Incompatible Predicates and Colours*, pages 97-102. CCSI/SCEIO, Saskatoon, Saskatchewan, 1982.
- Parsons, Terence. *Nonexistent Objects*. New Haven:Yale University Press, 1980.
- Patil, R.S. *Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis*. Technical Report MIT/LCS/TR-267, Laboratory for Computer Science, Massachusetts Institute of Technology, 1981. PhD Thesis.
- Patil, R., Szolovits, P., Schwartz, W. Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders. In P. Szolovits (Ed.), *Artificial Intelligence in Medicine*. Westview Press, 1982.
- Pereira, F., and Warren, D. Definite Clause Grammars for Language Analysis. *Artificial Intelligence*, 1980, 13(3), 231-278.
- Poole, D. *A Logical System for Default Reasoning*, pages 373-384. Proceedings of the AAAI Workshop on Nonmonotonic Reasoning, New Paltz, New York, 1984.
- Poole, D. *On the Comparison of Theories: Preferring the Most Specific Explanation*, pages 144-147. IJCAI 85, Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, California, 1985.
- Poole, D., and Goebel, R. On Eliminating Loops in Prolog. *ACM SIGPLAN Notices*, 1985, 20(8), 38-41.
- Pople, H. *The Formation of Composite Hypotheses in Diagnostic Problem Solving - An Exercise in Synthetic Reasoning*

- [*INTERNIST*], pages 1030-1037. IJCAI77. Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, Massachusetts. 1977.
- Pople, H. Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics. In P. Szolovits (Ed.), *Artificial Intelligence in Medicine*, Westview Press. 1982.
- Popper, K. *The Logic of Scientific Discovery*. New York, New York:Harper and Row. 1958.
- Putnam, H. The Meaning of 'Meaning'. In Putnam, H. (Ed.), *Mind, Language, and Reality*. Cambridge, England: Cambridge University Press. 1975.
- Pylyshyn, Z. What the Mind's Eye Tells the Mind's Brain: A Critique of Mental Imagery. *Psychological Bulletin*, 1973, 80(1), 1-24.
- Pylyshyn Z. The Imagery Debate: Analog Media versus Tacit Knowledge. In Block, N. (Ed.), *Imagery*, Cambridge, Ma.: M.I.T. Press. 1981.
- Quillian, M. Semantic Memory. In Minsky, M. (Ed.), *Semantic Information Processing*, Cambridge, Ma.: M.I.T. Press. 1968.
- Quillian, M. The Teachable Language Comprehender. *Communications of the ACM*, 1969, 12(8), 459-476.
- Quine, W. *Word and Object*. Cambridge, Ma.:M.I.T. Press. 1960.
- Quine, W., and Ullian, J. *The Web of Belief*. New York, New York:Random House. 1978.
- Rapaport, William J. *Intentionality and the Structure of Existence*. PhD thesis, Indiana University Department of Philosophy. 1976.
- Rapaport, William J. Meinongian Theories and a Russellian Paradox. *Noûs*, 1978, 12, 153-80. errata, *Noûs* 13(1979)125.

- Rapaport, William J. How to Make the World Fit Our Language: An Essay in Meinongian Semantics. *Grazer Philosophische Studien*. 1981. 14. 1-21.
- Rapaport, William J. Meinong, Defective Objects, and (Psycho-)Logical Paradox. *Grazer Philosophische Studien*. 1982. 18. 17-39.
- Rapaport, William J. Critical Notice of Routley 1979. *Philosophy and Phenomenological Research*. 1984. 44. 539-52.
- Rapaport, William J. *Belief Representation and Quasi-Indicators*. Technical Report 215. SUNY Buffalo Department of Computer Science. 1984.
- Rapaport, William J. *Meinongian Semantics for Propositional Semantic Networks*, pages 43-48. ACL. 1985.
- Rapaport, William J. To Be and Not to Be. *Noûs*. 1985. 19. 255-71.
- Rapaport, William J. Review of Lambert 1983. *Journal of Symbolic Logic*. 1986. 51. 248-52.
- Rapaport, William J. Logical Foundations of Belief Representation. *Cognitive Science*. 1986. 10. 00-00.
- Rapaport, William J., and Shapiro, Stuart C. *Quasi-Indexical Reference in Propositional Semantic Networks*, pages 65-70. ACL. Stanford University. 1984.
- Raphael, B. SIR: Semantic Information Retrieval. In Minsky, M. (Ed.). *Semantic Information Processing*, Cambridge, Mass.: MIT Press. 1968.
- Reghbati, H.K. *VLSI Testing and Validation Techniques*. IEEE Computer Society Press. 1985.
- Reiter, R. A Logic for Default Reasoning. *Artificial Intelligence*. 1980. 13(1). 81-132.
- Reiter, R., and Crisculo, G. Some Representational Issues in Default Reasoning. In Cercone, N. (Ed.), *Computational*

Linguistics. London: Pergamon Press. 1983.

Rich. C. *A Formal Representation for Plans in the Programmer's Apprentice*. IJCAI. Vancouver, British Columbia. 1981.

Rich. C. *A layered architecture of a system for reasoning about programs*, pages 540-546. IJCAI. Los Angeles, California. 1985.

Rieger. C., Grinberg. M. *The Causal Representation and Simulation of Physical Mechanics*. Technical Report TR-495. University of Maryland. November 1976.

Roberts. L. Machine Perception of Three-Dimensional Objects. In J. Tippet (Eds.), *Optical and Electro-Optical Information Processing*, Cambridge, Massachusetts: MIT Press. 1965.

Roberts. R. and Goldstein. I. *The FRL Primer*. Technical Report AI Memo 408. MIT. November 1977.

Robinson. J. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 1965, 12, 23-41.

Robinson. J. *Logic: Form and Function*. Edinburgh, Scotland:Edinburgh University Press. 1979.

Robson. D. Object-Oriented Software Systems. *Byte*, 1981, 6(8), 74-86.

Rosenschein. S., and Shieber. M. *Translating English into Logical Form*, pages 1-8. Proceedings of the 20th ACL Conference, Toronto, Ontario, 1982.

Roussel. P. *PROLOG: Manuel de Reference et d'Utilization*. Technical Report , Aix-Marseille University. 1975.

Routley. Richard. *Exploring Meinong's Jungle and Beyond*. Technical Report. Australian National University, Research School of Social Sciences, Department of Philosophy. 1979.

Rumelhart. D., and Ortony. A. *The Representation of Knowledge in Memory*. Technical Report TR 55. Department of Psychology. University of California. San Diego. 1976.

- Sacerdoti, E.D. *A structure for plans and behavior*. Technical Report 109, SRI Artificial Intelligence Center, 1975.
- Samad, T. and Director, S.W. Natural Language Interface for CAD: A First Step. *IEEE Design and Test*, August 1985, 2(4), 78-86.
- Sandewall, E. A Functional Approach to Non-Monotonic Logic. *Computational Intelligence*, 1985, 1(2), 69-81.
- Schank, R. Conceptual Dependency: A Theory of Natural Language Understanding. *Cognitive Psychology*, 1972, 3, 552-631.
- Schank, R. The Role of Memory in Language Processing. In C. Cofer (Ed.), *The Structure of Human Memory*, San Francisco: Freeman, 1975.
- Schank, R., and Abelson, R. *Scripts, Plans, and Knowledge*, pages 151-157. IJCAI75, Advance Papers of the 4th International Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975.
- Schank, R., and Abelson, R. *Scripts, Plans, and Understanding*. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1977.
- Schank, R., and Rieger, C. Inference and the Computer Understanding of Natural Language. *Artificial Intelligence*, 1974, 5(4), 373-412.
- Schank, R., and Wilensky, R. Response to Drescher and Hornstein. *Cognition*, 1977, 5, 133-146.
- Schank, R., Goldman, N., Rieger, C., and Riesbeck, C. *Margie: Memory, Analysis, Response Generation and Inference on English*, pages 255-261. IJCAI73, Proceedings of the 3rd International Joint Conference on Artificial Intelligence, Stanford, Ca., 1973.
- Schlipf, J. Private Communication. . 1986. (). .
- Schubert, L. *Extending the Expressive Power of Semantic Networks*, pages 158-164. IJCAI75, Advance Papers of the 4th International Joint Conference on Artificial

Intelligence. Tbilisi, USSR, 1975.

Schubert, L. Extending the Expressive Power of Semantic Networks. *Artificial Intelligence*, 1976, 7, 163-198.

Schubert, L. *Problems with Parts*, pages 778-784. IJCAI, Tokyo, Japan, 1979.

Schubert, L. *An approach to the syntax and semantics of affixes in 'conventionalized' phrase structure grammar*, pages 189-195. CSCSI/SCEIO, Saskatoon, Saskatchewan, 1982.

Schubert, L. *On Parsing Preferences*, pages 247-250. Coling, Stanford, California, 1984.

Schubert, L., and Pelletier, F. From English to Logic: context-free computation of conventional logical translations. *Am. Journal of Computational Linguistics*, 1982, 8(1), 26-44.

Schubert, L., Goebel, R., and Cercone, N. The Structure and Organisation of a Semantic Network for Comprehension and Inference. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*, New York: Academic Press, 1979.

Scragg, G. *Answering Questions About Processes*, Ph.D. Thesis. Technical Report, Department of Computing Science, University of California, San Diego, 1975.

Shapiro, S. *A Net Structure for Semantic Information Storage, Deduction, and Retrieval*, pages 512-523. IJCAI71, Proceedings of the 2nd International Joint Conference on Artificial Intelligence, London, England, 1971.

Shapiro, Stuart C. *The MIND System: A Data Structure for Semantic Information Processing*. Technical Report Report Number R-837-PR, The Rand Corporation, 1971. also AD Number 733 560, Defense Documentation Center, Alexandria, VA.

Shapiro, Stuart C. *A Net Structure for Semantic Information Storage, Deduction and Retrieval*, pages 512-23. IJCAI, 1971.

- Shapiro, Stuart C. *Representing and Locating Deduction Rules in a Semantic Network*, pages 14-18. IJCAI, 1977.
- Shapiro, Stuart C. Path-Based and Node-Based Inference in Semantic Networks. In Waltz, D. (Ed.), *Tinlap-2: Theoretical Issues in Natural Language Processing*, New York: ACM, 1978.
- Shapiro, S. The SNePS Semantic Network Processing System. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*, New York: Academic Press, 1979.
- Shapiro, Stuart C. The SNePs Semantic Network Processing System. In Findler, N.V. (Ed.), *Associative Networks: The Representation and Use of Knowledge by Computers*, New York: Academic Press, 1979.
- Shapiro, Stuart C. *Numerical Quantifiers and Their Use in Reasoning with Negative Information*, pages 791-96. IJCAI, 1979.
- Shapiro, Stuart C. Generalized Augmented Transition Network Grammars for Generation from Semantic Networks. *American Journal of Computational Linguistics*, January-March, 1982, 8.1, 12-25.
- Shapiro, E. *A Subset of Concurrent Prolog and its Interpreter*. Technical Report CS83-06, Weizmann Institute of Science, 1983.
- Shapiro, Stuart C.; and McKay, Donald P. *Inference with Recursive Rules*, pages 151-53. AAAI, 1980.
- Shapiro, Stuart C., and Wand, Mitchell. *The Relevance of Relevance*. Technical Report 46, Indiana University Department of Computer Science, 1976.
- Shapiro, Stuart C., and Woodmansee, G. H. *A Net Structure Based Relational Questions Answerer: Description and Examples*, pages 325-46. IJCAI, 1969.
- Shapiro, Stuart C.; Martins, João; and McKay, Donald P.

Bi-Directional Inference. pages 90-93. Cognitive Science Society, 1982.

- Shapiro, Stuart C.; McKay, Donald P. Martins, João; and Morgado, Ernesto. *SNePSLOG: A 'Higher Order' Logic Programming Language*. Technical Report SNeRG Technical Note Number 8, SUNY Buffalo Department of Computer Science, 1981. Presented at the 1981 Workshop on Logic Programming for Intelligent Systems, Long Beach, California.
- Shapiro, Stuart C.; Srihari, Sargur N.; Geller, James; and Taie, Ming-Ruey. A Fault Diagnosis System Based on an Integrated Knowledge Base. *IEEE Software*, March 1986, 3.2, 48-49.
- Shapiro, Stuart C.; Woodmansee, G. H.; and Kreuger, M. W. A *Semantic Associational Memory Net that Learns and Answers Questions (SAMENLAQ)*. Technical Report 8, University of Wisconsin Computer Sciences Department, 1968.
- Shepard, R. The Mental Image. *American Psychologist*, 1978, 33(6), 125-137.
- Shepard, R., and Cooper, L. *Mental Images and Their Transformations*. Cambridge, Mass.:MIT Press, 1982.
- Shibahara, T. *On Using Causal Knowledge to Recognize Vital Signals: Knowledge-based Interpretation of Arrhythmias*. IJCAI, Los Angeles, California, 1985.
- Shibahara, T., Tsotsos, J., Mylopoulos, J., Covey, H. *CAA: A Knowledge-Based System Using Causal Knowledge to Diagnose Cardiac Rhythm Disorders*. IJCAI, Karlsruhe, W. Germany, August, 1983.
- Shortliffe, E. *Computer - Based Medical Consultation: MYCIN*. New York:North-Holland, 1976.
- Simon, H. Rational Choice and the Structure of the Environment. *Psychological Review*, 1956, 63(1), 129.

- Slager, C., et al. *Left Ventricular Contour Segmentation from Anatomical Landmark Trajectories and its Application to Wall Motion Analysis*. Computers in Cardiology, Geneva, 1979.
- Sleeman, D., and Brown, J. (eds). *Intelligent Tutoring Systems*. New York, New York:Academic Press, 1982.
- Sloman, A. *Afterthoughts on Analogical Reasoning*, pages 178-182. Proceedings Theoretical Issues on Natural Language Processing, Cambridge, Massachusetts, 1975.
- Smith, B. *Reflections and Perspectives in a Procedural Language*. Technical Report TR-272, M.I.T., 1982.
- Sowa, M., Scott, A., and Shortliffe, E. Completeness and Consistency in Rule Based Systems. In Buchanan, B., and Shortliffe, E. (Eds.), *Rule Based Expert Systems*, Reading, Mass.: Addison-Wesley, 1985.
- Srihari, Rohini K. *Combining Path-Based and Node-Based Inference in SNePS*. Technical Report 183, SUNY Buffalo Department of Computer Science, 1981.
- Srihari, Sargur N.; Hull, Jonathan J.; Palumbo, Paul W.; Niyogi, Debashish; and Wang, Ching-Huei. *Address Recognition Techniques in Mail Sorting: Research Directions*. Technical Report 85-09, SUNY Buffalo Department of Computer Science, 1985.
- Stefik, M. et al. *The Partitioning of Concerns in Digital System Design*, pages 43-52. IEEE, 1982.
- Steinberg, L.I. and Mitchell, T.M. The Redesign System: A Knowledge-Based Approach to VLSI CAD. *IEEE Design and Test*, February 1985, 2(1), 45-54.
- Stickel, M. *Theory Resolution: Building in Nonequational Theories*, pages 391-397. AAAI, Washington, D.C., 1983.
- Stickel, M. *Automated Deduction by Theory Resolution*, pages 1181-1186. IJCAI, Los Angeles, California, 1985.
- Suchin, Jennifer. *A Semantic Network Representation of the*

Peripheral Nervous System. Technical Report Project Report, SUNY Buffalo Department of Computer Science, 1985.

Sussman, Gerald J. *A Computer Model of Skill Acquisition.* American Elsevier Publishing Company, Inc., 1975.

Sussman, G.J. and Stallman, R. Heuristic Techniques in Computer-Aided Circuit Analysis. *IEEE Transactions on Circuits and Systems*, November 1975. CAS-22(11), 205-280.

Sussman G.J. and Steele, G.L. CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence*, August 1980, 14(1), 1-39.

Szolovits, P. and Pauker, S.G. Categorical and Probabilistic Reasoning in Medical Diagnosis. *Artificial Intelligence*, 1978, 11, 115-144.

Tate, Austin. *Generating Project Networks.* IJCAI, AAAI, 1977.

Taugher, J. *A Representation for Time Information.* Technical Report M.Sc. thesis, Department of Computing Science, University of Alberta, 1983.

Taugher, J., and Schubert, L. Fast Temporal Inference. 1986. (), (to appear).

Tenenberg, J. *Taxonomic Reasoning.* pages 191-193. IJCAI, Los Angeles, California, 1985.

Tomberlin, James E. *Agent, Language, and the Structure of the World.* Indianapolis:Hackett, 1984. R.A. George, translator.

Tranchell, Lynn M. *A SNePS Implementation of KL-ONE.* Technical Report 198, SUNY Buffalo Department of Computer Science, 1982.

Tsotsos, J. *Representational Axes and Temporal Cooperative Processes.* Technical Report RBCV-TR-2, Department of Computer Science, University of Toronto, May 1984.

- Tsotsos. J.K. Knowledge Organization and Its Role in Representation and Interpretation for Time-Varying Data: The ALVEN System. *Computational Intelligence*. 1985. 1(1).
- Tsotsos J.K., Mylopoulos. J., Covey. H.D., Zucker. S.W. A Framework For Visual Motion Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. November 1980. 1. .
- Ullman. J. *Principles of Database Systems*. New York:Computer Science Press. 1984.
- Umrigar. Z., and Pitchumani. V. An Experiment in Programming with Full First Order Logic. *IEEE 1985 Symposium on Logic Programming*. 1985. 1(1). 40-47.
- van Emden. M. Programming with Resolution Logic. In Elcock. E. and Michie. D. (Ed.). *Machine Intelligence 8*. Chichester. UK: Ellis Horwood. 1977.
- van Emden. M., and Kowalski. R. The semantics of predicate logic as a programming language. *Journal of the ACM*. 1976. 23(4). 733-742.
- van Melle. W. *A Domain Independent System that Aids in Constructing Consultation Programs (EMYCIN)*. Technical Report STAN-CS-80-820, Department of Computer Science. Stanford University. 1980.
- Vere. Steven. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1983. (PAMI-5/3). 246-267.
- Vilain. M. *The restricted language architecture of a hybrid representation system*. pages 547-551. IJCAI, Los Angeles. California. 1985.
- Vilain. M., and Kautz. H. *Constraint Propagation Algorithms for Temporal Reasoning*. pages (to appear). AAAI. Philadelphia. Pennsylvania. 1986.

- Wallis, J., Shortliffe, E. *Explanatory Power for Medical Expert Systems: Studies in the Representation of Causal Relationships for Clinical Consultations*. Technical Report STAN-CS-82-923, Stanford University, 1982.
- Walther, C. *A many-sorted calculus based on resolution and paramodulation*, pages 882-891. IJCAI, Karlsruhe, West Germany, 1983.
- Waltz, D. Understanding Line Drawings of Scenes with Shadows. In P. Winston (Ed.), *Psychology of Computer Vision*, New York: McGraw Hill, 1975.
- Warren, D., and Pereira, S. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *American Journal of Computational Linguistics*, 1982, 8(3-4), 110-122.
- Warren, D., Pereira, L., and Pereira, F. PROLOG - the language and its implementation compared to LISP. *Sigart Newsletter*, 1977, 8(64), 109-115.
- Weiss, S. M., and Kulikowski, C. A. *A Practical Guide to Designing Expert Systems*. Totowa, New Jersey: Rowman and Allanheld, 1984.
- Weyhrauch, R. Prolegomena to a Theory of Mechanized Reasoning. *Artificial Intelligence*, 1980, 13(2), 133-170.
- Wilensky, R. *Understanding Goal-Based Stories*. Technical Report Research Report 140, Department of Computer Science, Yale University, 1978.
- Williams, B.C. Qualitative Analysis of MOS Circuits. *Artificial Intelligence*, 1984, 24, .
- Wilson, K. *From Association to Structure*. Amsterdam: North Holland, 1980.
- Winograd, T. *Understanding Natural Language*. New York: Academic Press, 1972.
- Winograd, T. Breaking the Complexity Barrier Again. *SIGPLAN Notices*, 1974, 9(6), x-y.

- Winograd. T. Frames and the Procedural-Declarative Controversy. In Bobrow. D., and Collins. A. (Eds.). *Representation and Understanding*, New York: Academic Press. 1975.
- Winograd. T. Towards a Procedural Understanding of Semantics. *Revue Internationale de Philosophie*. 1976. 3-4(1). 260-303.
- Winograd. T. On Some Contested Suppositions of Generative Linguistics about the Scientific Study of Language. *Cognition*. 1977. 5(3). 151-179.
- Winston. P. *Learning Structural Descriptions from Examples*. Technical Report MAC-TR-76. Project MAC. M.I.T.. 1970.
- Woods. W. *Semantics for a Question-Answering System*. Technical Report Ph.D. Thesis. Division of Engineering. Harvard University. 1967.
- Woods. W. What's in a Link. In Bobrow. D., and Collins. A. (Eds.). *Representation and Understanding*, New York: Academic Press. 1975.
- Woods. W. Cascaded ATN Grammars. *American Journal of Computational Linguistics*, 1980. 6(1). 1-15.
- Woods. W. What's Important about Knowledge Representation. In McCalla. G., and Cercone. N. (Eds.). *IEEE Computer special issue on Knowledge Representation*, New York: IEEE. 1983.
- Woods. W. Problems in Procedural Semantics. In Pylyshyn. Z., and Demopoulos. W. (Eds.). *Meaning and Cognitive Structure*, New Jersey: Ablex Publishing Company. 1986.
- Woods. W., Kaplan. R., and Nash-Webber. B. *The Lunar Science Natural Language Report*. Technical Report TR2378. Bolt Beranek and Newman, Inc.. 1972.
- Wright. M., and Fox. M. *SRL/15 User Manual*. Technical Report TR. Carnegie Mellon University. 1982.
- Wygralak. M. Fuzzy Cardinals based on the Generalized

Equality of Fuzzy Subsets. *Fuzzy Sets and Systems* 18, 1986, 1, 143-158.

Xiang, Zhigang, and Srihari, Sargur N. *Spatial Structure and Function Representation in Diagnostic Expert Systems*, pages 191-206. IEEE, 1985.

Xiang, Zhigang; Srihari, Sargur N.; Shapiro, Stuart C.; and Chutkow, Jerry G. *Analogical and Propositional Representations of Structure in Neurological Diagnosis*, pages 127-32. IEEE, Silver Spring, MD., 1984.

Zadeh, L.A. Similarity Relations and Fuzzy Orderings. *Inf. Sci.*, 1971, 3, 177-200.

Zadeh, L.A. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. *Information Science*, 1975, 8, 199-249, 301-357.

Zadeh, L.A. *A Theory of Approximate Reasoning*. Technical Report Memorandum M77/58. Electronics Research Laboratory, 1977. Also appears in *Machine Intelligence 9*, Hayes, J.E., Michie, D., and Kulich, L.I. (eds.) New York: Wiley, pp. 149-194, 1979.

Zadeh, L.A. *A Theory of Approximate Reasoning*. Technical Report Memorandum M77/58. Electronics Research Laboratory, 1977. also appears in *Machine Intelligence 9*, Hayes, J.E., Michie, D., and Kulich, L.I. (eds.) New York: Wiley, pp. 149-194, 1979.

Zadeh, L.A. PRUF - A Meaning Representation Language for Natural Languages. *Int. J. Man-Machine Studies*, 1978, 10, 395-460.

Zadeh, L. A Theory of Approximate Reasoning. *Machine Intelligence*, 1979, 9, 149-194.

Zadeh, L.A. *Possibility Theory and Soft Data Analysis*. Technical Report Memorandum M79/59. Electronics Research Laboratory, 1979. also appears in *Mathematical Frontiers of the Social and Policy Sciences*, Cobb, L. and Thrall, R.M. (eds.), Boulder: Westview Press, pp. 69-129.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.